Windows 8.1, Windows Server 2012 & Windows Server 2012 R2

The Personal Trainer™

Windows Command Line

William Stanek

Award-winning technology expert

Windows Command-Line:

The Personal Trainer

Windows 8.1, Windows Server 2012 & Windows Server 2012 R2

William Stanek

PUBLISHED BY

Stanek & Associates
PO Box 362
East Olympia, WA 98540-0362

Cover Design: Creative Designs Ltd.
Editorial Development: Andover Publishing Solutions
Technical Review: L & L Technical Content Services

You can provide feedback related to this book by emailing the author at williamstanek@aol.com. Please use the name of the book as the subject line.

Contents at a Glance

Table of Contents

Introduction

Windows Command Line: The Personal Trainer is the authoritative quick reference guide to Windows Command Line and is designed to be a key resource you turn to whenever you have questions about Windows Command Line. To this end, the book zeroes in on the key aspects of Windows Command Line that you'll use the most.

Inside this book's pages, you'll find comprehensive overviews, step-by-step procedures, frequently used tasks, documented examples, and much more. One of the goals is to keep the content so concise that the book remains compact and easy to navigate while at the same time ensuring that the book is packed with as much information as possible—making it a valuable resource.

What's This Book About?

In this book, I teach you how Windows Command Line features work, why they work the way they do, and how to use them to meet your needs. One of the goals is to keep the content so concise that the book remains compact and easy to navigate while ensuring that the book is packed with as much information as possible—making it a valuable resource. In addition, this book provides hands-on, tips, and best practices for working with Windows Command Line.

What Do I Need to Know?

To get practical and useful information into your hands without the clutter of a ton of background material, I had to assume several things. If you are reading this book, I hope that you have basic networking skills and a basic understanding of Windows. With this in mind, I don't devote entire chapters to understanding Windows architecture, installing Windows, or Windows startup and shutdown. I also assume that you are fairly familiar with Windows commands and procedures as well as the Windows user interface. If you need help learning Windows basics, you should read the Windows documentation.

How Is This Book Organized?

Rome wasn't built in a day, and this book wasn't intended to be read in a day, a week, or even 21 days. Ideally, you'll read this book at your own pace, a little each day, as you work your way through the features Windows Command Line has to offer.

Making this book easy to follow and understand was my number one goal. I really want anyone, skill level or work schedule aside, to be able to learn how to effectively use Windows Command Line.

To make the book easy to use, this book is organized into multiple chapters. The chapters are arranged in a logical order, taking you from the essential concepts you need to know to work with Windows Command Line to techniques for managing computers with scripts.

Windows Command Line: The Personal Trainer is designed to be used with *Windows Command Line for Administration: The Personal Trainer*. While this book focuses on core features, the latter book focuses using Windows Command Line to manage computers, networks, printers and Active Directory.

What Conventions Are Used in This Book?

I've used a variety of elements to help keep the text clear and easy to follow. You'll find code terms and listings in `monospace` type, except when I tell you to actually enter a command. In that case, the command appears in **bold** type. When I introduce and define a new term, I put it in *italics*.

This book also has notes, tips and other sidebar elements that provide additional details on points that need emphasis.

Other Resources

Although some books are offered as all-in-one guides, there's simply no way one book can do it all. This book is intended to be used as a concise and easy-to-use resource. It covers everything you need to perform core tasks with Windows Command Line, but it is by no means exhaustive.

As you encounter new topics, take the time to practice what you've learned and read about. Seek additional information as necessary to get the practical experience and knowledge that you need.

I truly hope you find that *Windows Command Line: The Personal Trainer* helps you use Windows Command Line successfully and effectively.

Thank you,

William R. Stanek

(williamstanek@aol.com)

Chapter 1. Overview of the Windows Command Line

The command line is built into the Microsoft Windows operating system and is accessed through the command-shell window. Every version of Windows has had a built-in command line, used to run built-in commands, utilities, and scripts. Although the command line is powerful and versatile, some never use it. If you are happy using the graphical tools, you may be able to use them forever without ever having to do anything more than point and click.

However, for proficient Windows users, skilled support staff, and IT professionals, the Windows command line is inescapable. Knowing how to use the command line properly—including which command-line tools to use when and how to work with the tools effectively—can mean the difference between smooth-running computers and frequent problems. And if you're responsible for multiple domains or networks, learning the time-saving strategies that the command line offers is not just important—it's essential for sustaining day-to-day operations.

In this chapter, I'll explain command line essentials, how to use built-in commands, how to run command-line utilities, and how to work with other support tools. Right up front I should tell you that Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 have many more command-line tools available by default than their predecessors. In fact, many of the tools that were previously available only when you installed the Windows Support Tools and the Windows Server Resource Kit tools are now available by default.

> *REAL WORLD* As you read this chapter, and the rest of the book, keep in mind that this book is written for Windows Server 2012, Windows Server 2012 R2 and Windows 8.1. Techniques that you learn in this book can be used on any of these operating systems unless otherwise noted. In some cases, you might be able to use the techniques discussed with other Windows operating systems, although the options or functions may vary. In any case, you should

always test commands, options, and scripts before using them. The best way to do this is in a development or test environment where the systems with which you are working are isolated from the rest of the network.

Command Line Essentials

Each new version of Windows has extended and enhanced the command line. The changes have been dramatic, and they've not only improved the performance capabilities of the command line but its versatility as well. Today you can do things with the Windows command line that you simply could not do in previous versions of Windows. To help you put the available options to use in the fastest, most productive manner, the discussion that follows explores command-shell options and configuration, in addition to providing tips for using the command history.

Understanding the Windows Command Shell

The most commonly used command line is the Windows command shell. The Windows command shell (Cmd.exe) is a 32-bit or 64-bit environment for working with the command line. On 32-bit versions of Windows, you'll find the 32-bit executable in the %SystemRoot%\System32 directory. On 64-bit versions of Windows, you'll find the 32-bit executable in the %SystemRoot%\SysWow64 directory and the 64-bit executable in the %SystemRoot%\System32 directory. Other command lines are available, such as Windows PowerShell (powershell.exe) discussed later in this chapter.

> *NOTE* %SystemRoot% refers to the *SystemRoot* environment variable. The Windows operating system has many environment variables, which are used to refer to user-specific and system-specific values. Often, I'll refer to environment variables using the standard Windows syntax %VariableName%.

You can start the command shell by using the Apps Search box. Enter cmd in the Apps Search box, and then press Enter. Or you can right-click the bottom left corner of the screen and then choose Command Prompt.

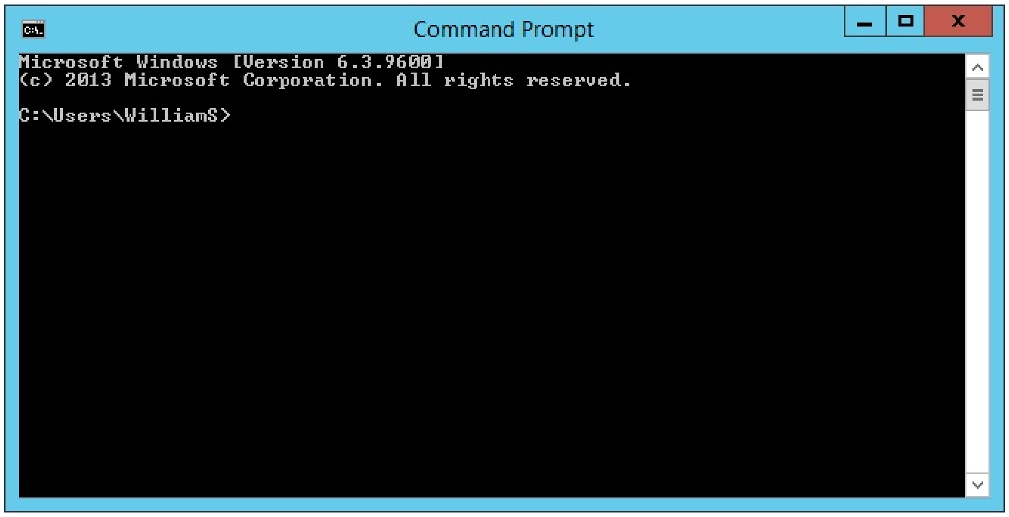
You can initialize the environment for the Windows command shell in several ways, including bypassing startup parameters to Cmd.exe or by using a custom startup file, which is placed in the %SystemRoot%\System32 directory. Figure 1-1 shows a command-shell window. By default, the command line is 80 characters wide and the command shell displays 25 lines of text. When additional text is to be displayed in the command-shell window or you enter commands and the command shell's window is full, the current text is displayed in the window and prior text is scrolled up. If you want to pause the display temporarily when a command is writing output, press Ctrl+S. Afterward, press Ctrl+S to resume or Ctrl+C to terminate execution.

> *NOTE* Custom startup files are used for MS-DOS programs that require special configurations. These files are named Autoexec.nt and Config.nt, and they are stored in the %SystemRoot%\System32 directory.

In this figure from Windows Server 2012 R2, the display text is

```
Microsoft Windows [Version 6.3.9600]
(C) Copyright 2013 Microsoft Corporation. All rights reserved.

C:\Users\williams>
```



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\WilliamS>
```

FIGURE 1-1 The command shell is the primary command-line window you'll use.

Here, the command prompt for the command line shows the current working directory, which by default is %UserProfile%, meaning the user profile directory for the current user. A blinking cursor following the command prompt indicates the command line is in interactive mode. In interactive mode, you can type commands directly after the prompt and press Enter to execute them. For example, type **dir** and then press Enter to get a listing of the current directory.

The command prompt also has a batch mode, which is used when executing a series of commands. In batch mode, the command prompt reads and executes commands one by one. Typically, batch commands are read from a script file, but batch commands can also be entered at the command prompt, such as when you use the FOR command to process each file in a set of files. (You'll learn more about batch scripts, loops, and command controls in Chapter 3, "Command-Line Scripting Essentials.")

Whenever you work with the Windows command line, it is important to keep in mind where the commands you are using come from. Native commands (commands built into the operating system by Microsoft) include:

- Internal commands that exist internally within the command shell and do not have separate executable files
- External commands that have their own executable files and are normally found in the %SystemRoot%\System32 directory

Table 1-1 shows a list of internal commands for the command shell (Cmd.exe). Each internal command is followed by a brief description.

TABLE 1-1 Quick Reference to Internal Commands for the Command Shell (Cmd.exe)

NAME	DESCRIPTION

assoc

Displays or modifies the current file extension associations.

call

Calls a procedure or another script from within a script.

cd (chdir)

Displays the current directory name or changes the location of the current directory.

cls

Clears the command window and erases the screen buffer.

color

Sets the text and background colors of the command-shell window.

copy

Copies files from one location to another or concatenates files.

date

Displays or sets the system date.

del (erase)

Deletes the specified file, files, or directory.

dir

Displays a list of subdirectories and files in the current or specified directory.

dpath

Allows programs to open data files in specified directories as if they were in the current directory.

echo

Displays text strings to the command line; sets command echoing state (on | off).

endlocal

Ends localization of variables.

exit

Exits the command shell.

for

Runs a specified command for each file in a set of files.

ftype

Displays current file types or modifies file types used in file extension associations.

goto

Directs the command interpreter to a labeled line in a batch script.

if

Performs conditional execution of commands.

md (mkdir)

Creates a subdirectory in the current or specified directory.

mklink

Creates either a symbolic or a hard link for either a file or a directory.

move

Moves a file or files from the current or designated source directory to a designated target directory. Can also be used to rename a directory.

path

Displays or sets the command path the operating system uses when searching for executables and scripts.

pause

Suspends processing of a batch file and waits for keyboard input.

popd

Makes the directory saved by PUSHD the current directory.

prompt

Sets the text for the command prompt.

pushd

Saves the current directory location and then optionally changes to the specified directory.

rd (rmdir)

Removes a directory or a directory and its subdirectories.

rem

Sets a remark in batch scripts or Config.sys.

ren (rename)

Renames a file or files.

set

Displays current environment variables or sets temporary variables for the current command shell.

setlocal

Marks the start of variable localization in batch scripts.

shift

Shifts the position of replaceable parameters in batch scripts.

start

Starts a separate window to run a specified program or command.

time

Displays or sets the system time.

title

Sets the title for the command-shell window.

type

Displays the contents of a text file.

verify

Causes the operating system to verify files after writing files to disk.

vol

Displays the disk’s volume label and serial number.

The syntax for using any internal (and most external commands) can be obtained by typing the command name followed by /? at the prompt, such as

```
copy /?
```

You’ll find there are many more external commands than internal commands, including ones that are very similar to those built into the command line. Most of these similar commands are extended or enhanced in some way. For example, the external XCOPY command is more versatile than the COPY command because it allows you to copy directory trees as well as files and offers many more parameters. With the external SETX command, you can write environment variable changes directly to the Windows registry, which makes the changes permanent rather than temporary as the SET command does.

> *TIP* SETX is one of many commands that now are available by default in both Windows 8.1 and Windows Server 2012 R2. You can also use SETX to obtain current Registry key values and write them to a text file.

Beyond this, the difference between internal and external commands isn’t very important. Many Windows utilities have command-line extensions that allow parameters to be passed to the utility from the command line, and thus are used like external commands.

Understanding Windows PowerShell

Windows PowerShell (powershell.exe) is a full-featured command shell that uses built-in commands called cmdlets and built-in programming features as well as standard command-line utilities. Windows PowerShell is included as part of Windows 8.1, Windows Server 2012 and Windows Server 2012 R2, and is available as a separate download for other operating systems. Because the included version of Windows PowerShell may not be the most recent, you may want to check the Microsoft Download Web site for a more recent version.

When you are working with Windows Server 2012 and Windows Server 2012 R2, a shortcut for starting Windows PowerShell is pinned to the desktop taskbar. You also can start the default version of Windows PowerShell using the Apps Search box. Type **powershell** in the Apps Search box, and then press Enter. Or, within another command line, type **powershell** and then press Enter. On 32-bit systems, this starts the 32-bit version of Windows PowerShell. On 64-bit systems, this starts the 64-bit version of Windows PowerShell. Both of which are stored by default in the %SystemRoot%\System32\WindowsPowerShell\Version directory where Version is the Version of PowerShell you installed, such as v1.0 or v2.0. On 64-bit versions of Windows, you'll find the 32-bit executable in the %SystemRoot%\SysWow64\WindowsPowerShell\Version directory.

When Windows PowerShell starts, you will see a message similar to the following:

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\williams>
```

You can disable this message by starting the shell with the –Nologo parameter, such as

```
powershell -nologo
```

> *NOTE* Regardless of how you start the shell, you know you are using Windows PowerShell because the command prompt title bar changes to Windows PowerShell or Command Prompt – powershell and the current path is preceded by *PS*.

You can start Windows PowerShell without loading profiles using the –Noprofile parameter, such as:

```
powershell -noprofile
```

The first time you start Windows PowerShell, you typically will see a message indicating that scripts are disabled and that none of the listed profiles is executed. This is the default secure configuration for Windows PowerShell. To enable scripts for execution, enter the following command at the shell prompt:

```
set-executionpolicy allsigned
```

This command sets the execution policy to require that all scripts have a trusted signature to execute. For a less restrictive environment, you can enter the following command:

```
set-executionpolicy remotesigned
```

This command sets the execution policy so that scripts downloaded from the Web execute only if they are signed by a trusted source, while allowing local scripts to run without requiring a digital signature. To work in an unrestricted environment, you can enter the following command:

```
set-executionpolicy unrestricted
```

This command sets the execution policy to run scripts regardless of whether they have a digital signature.

When you are working with Windows PowerShell, you can enter the name of the cmdlet at the prompt and it will run in much the same way as a command-line command. You can also execute cmdlets from within scripts. At the Windows PowerShell prompt, you can get a complete list of cmdlets available by typing **help -**.

Cmdlets are named using verb-noun pairs. The verb tells you what the cmdlet does in general. The noun tells you what specifically the cmdlet works with. For example, the Get-Service cmdlet either gets information on all Windows services or information on a specifically named Windows service. To get help documentation on a specific cmdlet, type **help** followed by the cmdlet name, such as **help get-service**.

All cmdlets have configurable aliases which act as execution shortcuts. To list all aliases available, type **get-item –path alias:** at the Windows PowerShell prompt. You can create an alias that invokes any command using the following syntax:

```
new-item –path alias:AliasName –value:FullCommandPath
```

where AliasName is the name of the alias to create and FullCommandPath is the full path to the command to run, such as:

```
new-item –path alias:sm –value:
c:\windows\system32\ServerManager.exe
```

This example creates the alias sm for starting the Server Manager administration tool available in Windows Server 2012 and 2012 R2. To use this alias, you would simply type **sm** followed by any required parameters and then press Enter.

Configuring Command-Line Properties

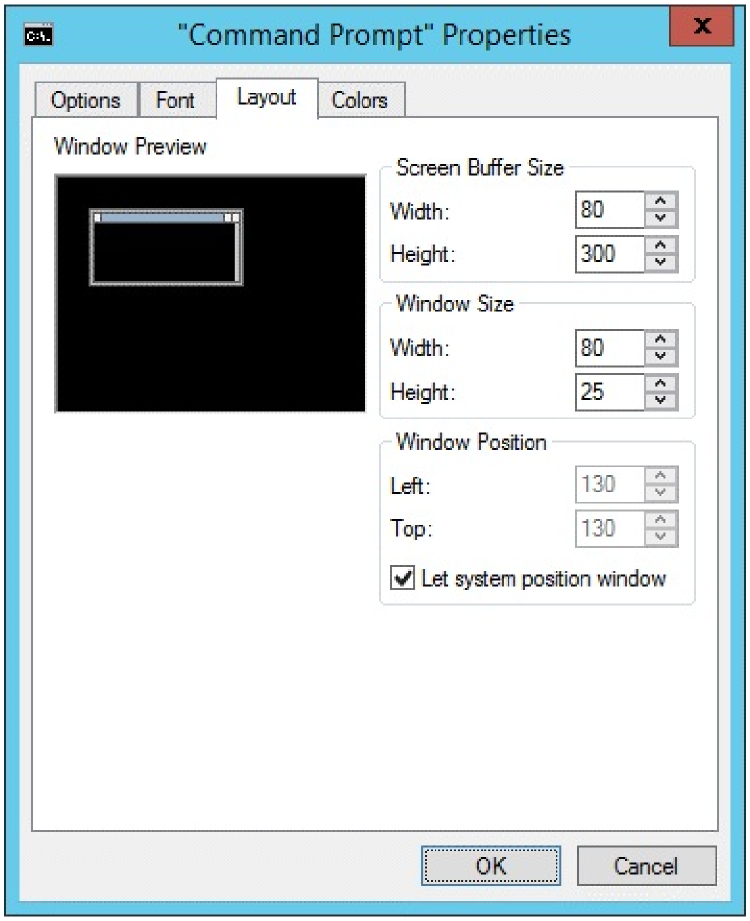
If you use the Windows command shell frequently, you'll definitely want to customize its properties. For example, you can add buffers so that text scrolled out of the viewing area is accessible. You can resize the command shell, change its fonts, and more.

To get started, click the command-prompt icon at the top of the command-shell window or right-click the console's display bar, and then select Properties. As Figure 1-2 shows, the Command Prompt Properties dialog box has four tabs:

- **Options** Allows you to configure cursor size, display options, edit options, and command history. Select QuickEdit Mode if you want to use the mouse to cut and paste text within the command window. Clear Insert Mode to use overwrite as the default editing mode. Use the command history to configure how previously used commands are buffered in memory. (You'll find more on the command history in the next section of this chapter, "Working with the Command History.")

TIP While working with text-only commands and tools, you may want to use Full Screen display mode to reduce the proportion of display space used by the command prompt itself. (To reset the display to command window mode, press Alt+Enter.) Afterward, type **exit** to exit the command prompt and return to the Windows desktop.

- **Font** Allows you to set the font size and face used by the command prompt. Raster font sizes are set according to their pixel width and height. For example, the size 8 x 12 is 8 screen pixels wide and 12 screen pixels high. Other fonts are set by point size, such as 10-point Lucida Console. Interestingly, when you select a point size of n, the font will be n pixels high; therefore, a 10-point font is 10 screen pixels high. These fonts can be designated as a bold font type as well, which increases their screen pixel width.
- **Layout** Allows you to set the screen buffer size, window size, and window position. Size the buffer height so that you can easily scroll back through previous listings and script output. A good setting is in the range of 1,000 to 2,000. Size the window height so that you can view more of the command-shell window at one time. A good setting is 45 lines on 800 x 600 screens with a 12-point font. If you want the command-prompt window to be in a specific screen position, clear Let System Position Window and then specify a position, in pixels, for the upper-left corner of the command window using Left and Top.
- **Colors** Allows you to set the text and background colors used by the command prompt. Screen Text and Screen Background control the respective color settings for the command-prompt window. The Popup Text and Popup Background options control the respective color settings for any popup dialog boxes generated when running commands at the command prompt.



Screen Buffer Size
Width: 80
Height: 300

Window Size
Width: 80
Height: 25

Window Position
Left: 130
Top: 130
☑ Let system position window

FIGURE 1-2 Configure the command-line properties for your environment.

When you are finished updating the command-shell properties, click OK to save your settings to your user profile. Your settings only modify the shortcut that started the current window. Any time you start a command line using the applicable shortcut, it will use these settings. If, however, you start a command line using a different shortcut, you'll have the settings associated with that shortcut.

Working with the Command History

The command history buffer is a feature of the Windows command shell (Cmd.exe) that remembers commands you've used in the current command-line session and allows you to access them without having to retype the command text. The maximum number of commands to buffer is set through the command-line Properties dialog box discussed in the previous section. By default, up to 50 commands are stored.

You can change the history size by completing these steps:

1. Right-click the command shell's title bar, select Properties, and then click the Options tab.
2. Use the Buffer Size field to set the maximum number of commands to store in the history and then click OK to save your settings to your user profile.

Your settings only modify the shortcut that started the current window. Any time you start a command line using the applicable shortcut, it will use these settings. If, however, you start a command line using a different shortcut, you'll have the settings associated with that shortcut.

You can access commands stored in the history in the following ways:

- **Browsing with the arrow keys** Use the up-arrow and down-arrow keys to move up and down through the list of buffered commands. When you find the command you want to use, press Enter to execute it as previously entered. Or you can modify the displayed command text by adding or changing parameters and then pressing Enter.
- **Browsing the command history pop-up window** Press F7 to display a pop-up window that contains a listing of buffered commands. Next, select a command using the arrow keys. (Alternatively, press F9, then press the corresponding number on the keyboard, and finally press Enter.) Execute the selected command by pressing Enter, or press Esc to close the pop-up window without executing a command.
- **Searching the command history** Enter the first few letters of the command you want to execute and then press F8. The command shell searches through the history for the first command that begins with the characters you've entered. Press Enter to execute it. Or, press F8 again to search the history

buffer for the next match in the command history.

As you work with the command history, keep in mind that each instance of Cmd.exe has its own set of command buffers. Thus, buffers are only valid in the related command shell context.

Making Supplemental Components Available

Microsoft designed Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 with an extensible component architecture. Because of this extensible architecture, Microsoft can make new components available for the operating systems through installer packages. Typically, Microsoft provides such files as individual Microsoft Update Standalone Package (.msu) files.

Installing and configuring new components is a two-part process. First you must register the components on the computer by installing the installer package. Then you must use an appropriate tool to configure the components. After you install a new component on Windows Server 2012 or 2012 R2, you use an appropriate wizard in Server Manager to install and configure the new role, role service, or feature. Once you install a new component on Windows 8.1, you use the Windows Features dialog box to install and configure the new feature. With Windows 8.1, one supplemental component that you'll want to install to enable remote administration is the Microsoft Remote Server Administration Tools for Windows 8.1.

Using the Microsoft Remote Server Administration Tools for Windows 8.1

The Microsoft Remote Server Administration Tools (RSAT) for Windows 8.1 is a collection of tools for remotely managing the roles and features in Windows Server 2012 or 2012 R2 from a computer running Windows 8.1. RSAT includes support for remote management of Windows Server 2012 or 2012 R2 regardless of whether the server is running a Server Core installation or a Full Server installation and provides similar functionality to the Windows Server 2012 Administration Tools Pack.

RSAT is available in a 32-bit edition and a 64-bit edition for computers running Windows 8.1. If your computer is running a 32-bit edition of Windows 8.1, you must install the 32-bit edition of RSAT. If your computer is running a 64-bit edition of Windows 8.1, you must install the 64-bit edition of RSAT. You can use the tools designed for either architecture to remotely manage 64-bit editions of Windows Server 2012 or 2012 R2.

You should not install the Microsoft Remote Server Administration Tools for Windows 8.1 on a computer that has the Windows Server 2008 Administration Tools Pack or the Windows Server 2003 Administration Tools Pack installed. You must remove all versions of earlier Administration Tools Pack tools from the computer before installing the Microsoft Remote Server Administration Tools for Windows 8.1.

Because the installer package for the remote administration tools is provided as an update, Microsoft assigns the installer package an identification number in the Microsoft Knowledge Base. Write down this number. If you need to uninstall or reinstall the installer package, this number will help you locate the update you need to uninstall or reinstall.

Registering the Remote Server Administration Tools Package

You can register the installer package for the Microsoft Remote Server Administration Tools for Windows 8.1 by completing the following steps:

1. Obtain the version of the Microsoft Remote Server Administration Tools for Windows 8.1 that is compatible with the architecture and service pack used on your Windows 8.1 computer. You can obtain the latest version of the tools for the latest Windows 8.1 service pack by visiting the Microsoft Download site (*http://download.microsoft.com/*).
2. If you've saved this file to a location on your computer or a network share, you can begin the installation process by accessing the file location in File Explorer and double-clicking the installer file.

3. When prompted to confirm the installation, click OK. When you are prompted, read the license terms. If you accept the terms, click I Accept to continue. The installer will then install the tools as an update for Windows 8.1.
4. When the installer completes the installation of the tools, click Close. Although the installer may not state this explicitly, you may need to restart your computer to finalize the installation.

Configuring and Selecting Remote Server Administration Tools

Registering the installer package ensures the remote administration tools are available for selection. You can configure and select the remote server administration tools that you want to use by completing the following steps:

1. Press Windows Key + I, click Control Panel, and then click Programs.
2. Under Programs And Features, click Turn Windows Features On Or Off.
3. In the Windows Features dialog box, expand Remote Server Administration Tools.
4. Using the options under the Feature Administration Tools and Role Administration Tools nodes, select the remote administration tools that you want to install and then click OK.

After you select the tools you want to use, Windows 8.1 automatically configures them for use. At the command line, you'll be able to use any command-line tools that Windows 8.1 has configured for use. The graphical versions of the tools are available on the Tools menu in Server Manager.

Removing the Remote Server Administration Tools

You can remove remote server administration tools that you no longer want to use by completing the following steps:

1. Press Windows Key + I, click Control Panel, and then click Programs.
2. Under Programs And Features, click Turn Windows Features On Or Off.
3. In the Windows Features dialog box, expand Remote Server Administration Tools.
4. Using the options under the Feature Administration Tools and Role Administration Tools nodes, clear the check boxes for any remote administration tool that you want to remove and then click OK.

Removing the Remote Server Administration Tools Package

If you no longer use a computer for remote administration and want to remove the remote administration tools completely, you can remove the entire installer package by completing the following steps:

1. Press Windows Key + I, click Control Panel, and then click Programs.
2. Under Programs And Features, click View Installed Updates.
3. Click the update you used to install the remote administration tools and then click Uninstall.
4. When prompted to confirm, click Yes.

Chapter 2. Getting the Most from the Command Line

The command shell provides a powerful environment for working with commands and scripts. As discussed in Chapter 1, “Overview of the Windows Command Line,” you can run many types of commands at the command line, including built-in commands, Windows utilities, and applications with command-line extensions. Regardless of its source, every command you’ll use follows the same syntax rules. These rules state that a command consists of a command name followed by any required or optional arguments. Arguments can also use redirection to specify the sources for inputs, outputs, and errors.

When you execute a command in the command shell, you start a series of events that are similar to the following:

1. The command shell replaces any variables you’ve entered in the command text with their actual values.
2. Multiple commands that are chained or grouped and passed on a single line are broken into individual commands and separated into command name and related arguments. The individual commands are then processed.
3. If the command name has a file path, the command shell uses this path to find the command. If the command cannot be found in the specified location, the command shell returns an error.
4. If the command name doesn’t specify a file path, the command shell tries to resolve the command name internally. A match means that you’ve referenced a built-in command that can be executed immediately. If no match is found, the command shell looks in the current directory for the command executable, and then searches the command path for the command executable. If the command cannot be found in any of those locations, the command shell returns an error.
5. If the command is located, the command is executed using any specified arguments, including those that specify the inputs to use.

Command output and any errors are written to the command window or to the specified destinations for output and error.

As you can see, many factors can affect command execution, including command path settings, redirection techniques used, and whether commands are chained or grouped. In this chapter, we'll use this breakdown of command execution to help you get the most out of the command shell. Before we dive into those discussions, however, let's look at special considerations for starting the command shell and introduce the concept of nesting command shells.

Managing Command Shell Startup

When you previously worked with the command line, you probably started a command prompt by typing **cmd** in the Apps Search box, and then choosing Command Prompt. However, because this technique starts the command prompt with standard user privileges rather than administrator privileges, you'll find that you are unable to perform many administrative tasks. To start the command prompt with administrator privileges, you need to type **cmd** in the Apps Search box, right-click Command Prompt, and then select Run As Administrator.

Another way to start a command line is to use the hidden shortcut menu. By default with Windows 8.1, Windows Server 2012 and Windows Server 2012 R2, Command Prompt and Command Prompt (Admin) are options on the shortcut menu that is displayed when you right-click in the lower left corner or press Windows key + X. Note that the alternative is for the Windows PowerShell prompt and the Windows PowerShell (Admin) prompt to be displayed on this menu. To configure which options are available, on the desktop, right-click the taskbar and then select Properties.

TIP Want to pin a Command Prompt shortcut to the taskbar? Type **cmd** in the Apps Search box, right-click Command Prompt, and then select Pin To Taskbar. Once you pin a shortcut to the taskbar, you can edit the shortcut properties to customize how the

Command Prompt starts.

You also can start a Command Prompt from within another Command Prompt or from a PowerShell prompt simply by entering **cmd**. Starting the Command Prompt in this way allows you to pass arguments to the command line, including switches that control how the command line works as well as parameters that execute additional commands. For example, you can start the command shell in quiet mode (meaning command echo is turned off) by using the startup command **cmd /q**. If you wanted the command shell to execute a command and then terminate, you could type **cmd /c** followed by the command text enclosed in quotation marks. The following example starts a command shell, sends the output of ipconfig to a file in a subdirectory named data, and then exits the command shell:

```
cmd /c "ipconfig > c:\data\ipconfig.txt"
```

NOTE The data subdirectory must exist for this command to work. Also note that when you start a command prompt from the Apps Search box, the command prompt runs with standard user privileges by default. With standard use privileges, you will not be able to perform certain administrator tasks or write data to secure system locations. For example, if you were to direct the output to the root of the C drive by typing **c:\ipconfig.txt**, the command prompt would not have sufficient privileges to create the file.

Table 2-1 summarizes the key parameters for the Windows command shell (Cmd.exe). Note that several command-line parameters are set by default. Because of this, the command line normally uses standard ANSI character codes for command output, as opposed to Unicode character codes, and enables command extensions that add features to most built-in commands.

TABLE 2-1 Essential Parameters for the Command Line

PARAMETER	DESCRIPTION
/C	

Executes the command specified and then exits the command shell.

/K

Executes the command specified and then remains in interactive mode.

/A

Command output to files (or pipes) is set to ANSI format (default).

/U

Command output to files (or pipes) is set to Unicode.

/Q

Turns on quiet mode, meaning command echo is off. By default, command echo is on.

/T:fg

Sets the foreground and background colors for the console window, where fg are the two values defined in the COLOR command.

/E:ON

Enables command extensions, which is the default.

/E:OFF

Disables command extensions.

NOTE Some parameters cannot be used with other switches. For example, you can't enable both Unicode and ANSI character codes. If you use both /A and *U, or* E:ON and /E:OFF, the command line applies the last option you passed on the command line.

Sometimes you may want to use different environment settings or parameters for a command line and then go back to your original settings without exiting the console window. To do this, you can use a technique called nesting. With nesting, you start a command line within a command line, and the nested command line inherits its environment settings from the current command line. You can then modify the environment as necessary and execute commands and scripts using those settings. When you type exit to end the nested command-line instance, you return to the previous command line and the previous environment settings are restored.

> *TIP* As you set out to work with the command shell, keep in mind that some characters have special meanings and that whenever the command shell encounters one of these characters, it attempts to carry out the special procedure associated with that character. Special characters include < > ( ) & | @ ^. If you want to use a special character as a regular character, you must escape the special character for the command shell to look at it literally, without invoking the special procedures with which it is associated. The escape character is the caret (^), which is the character above the 6 key on a standard keyboard, and is placed to immediately precede the special character.

Working with the Command Path

The Windows operating system uses the command path to locate executables. The types of files that Windows considers to be executables are determined by the file extensions for executables. You can also map file extensions to specific applications by using file associations. The two sections that follow discuss techniques for working with the command path, file extensions, and file associations.

Managing the Command Path

You can view the current command path for executables by using the PATH command. Start a command shell, type **path** on a line by itself, and press Enter. If you've installed Windows PowerShell, the results should look similar to the following:

```
PATH=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;
C:\Windows\System32\PowerShell\v2.0
```

> *NOTE* Observe the use of the semicolon (;) to separate individual paths. The command shell uses the semicolon to determine where one file path ends and another begins.

The command path is set during logon using system and user environment variables, namely the %PATH% variable. The order in which directories are listed in the path indicates the search order used by the command line when looking for executables. In the previous example, the command line searches in this order:

1. C:\Windows\system32
2. C:\Windows
3. C:\Windows\System32\Wbem
4. C:\Windows\System32\PowerShell\v1.0

You can permanently change the command path in the system environment using the SETX command. For example, if you use specific directories for scripts or applications, you may want to update the path information. You can do this by using the SETX command to add a specific path to the existing path, such as **setx PATH "%PATH%;C:\Scripts"**.

> *NOTE* Observe the use of the quotation marks and the semicolon (;). The quotation marks are necessary to ensure that the value *%PATH%*;C:\Scripts is read as the second argument for the SETX command. And, as discussed previously, the semicolon is used to specify where one file path ends and another begins.
>
> *TIP* Because the command path is set during logon, you must log off and then log on to see the revised path in your command prompts. If you'd rather not log off, you can verify that you've set the command path properly using the System Properties dialog box. In Control Panel\System, click Advanced System Settings in the Tasks Pane and then click Environment Variables on the Advanced tab of the System Properties dialog box.

In this example, the directory C:\Scripts is appended to the existing command path, and the sample path listed previously would be modified to read as follows:

```
PATH=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;
C:\Windows\System32\PowerShell\v1.0;C:\Scripts
```

Don’t forget about the search order that Windows uses. Because the paths are searched in order, the C:\Scripts directory will be the last one searched. This can sometimes slow the execution of your scripts. To help Windows find your scripts faster, you may want C:\Scripts to be the first directory searched. In this case, you could set the command path using the following command:

```
setx PATH "C:\Scripts;%PATH%"
```

Be careful when setting the command path. It is easy to overwrite all path information accidentally. For example, if you don’t specify the %PATH% environment variable when setting the path, you will delete all other path information. One way to ensure that you can easily re-create the command path is to keep a copy of the command path in a file. To write the current command path to a file, type **path > orig_path.txt**. Keep in mind that if you are using a standard command prompt rather than an administrator command prompt, you won’t be able to write to secure system locations. In this case, you can write to a subdirectory to which you have access or your personal profile. To write the command path to the command-shell window, type **path**.

Now you have a listing or a file that contains a listing of the original command path. Not only does the path command list the current command path, it also can be used to set the command path temporarily for the current command shell. For example, type **path %PATH%;C:\Scripts** to append the C:\Scripts directory to the command path in the current command shell.

Managing File Extensions and File Associations

File extensions are what allow you to execute commands by typing just their command name at the command line. Two types of file extensions are used:

- **File extensions for executables** Executable files are defined with the *%PATHEXT%* environment variable. You can view the current settings by typing set pathext at the command line. The default setting is PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MS With this setting, the command line knows which files are executable and which files are not, so you don’t have to specify the file extension at the command line.

- **File extensions for applications** File extensions for applications are referred to as *file associations*. File associations are what enable you to pass arguments to executables and to open documents, spreadsheets, or other application files by double-clicking their file icons. Each known extension on a system has a file association that you can view by typing **assoc** followed by the extension, such as **assoc .exe**. Each file association in turn specifies the file type for the file extension. This can be viewed by typing the FTYPE command followed by the file association, such as **ftype exefile**.

With executables, the order of file extensions sets the search order used by the command line on a per-directory basis. Thus, if a particular directory in the command path has multiple executables that match the command name provided, a .com file would be executed before a .exe file and so on.

Every known file extension on a system has a corresponding file association and file type—even extensions for executables. In most cases, the file type is the extension text without the period followed by the keyword file, such as cmdfile, exefile, or batfile, and the file association specifies that the first parameter passed is the command name and that other parameters should be passed on to the application.

You can look up the file type and file association for known extensions using the ASSOC and FTYPE commands. To find the association, type assoc followed by the file extension that includes the period. The output of the ASSOC command is the file type. So if you type **ftype *association*** (where *association* is the output of the ASSOC command), you’ll see the file type mapping. For example, if you type the following command to see the file associations for .exe executables: **assoc .exe**, you then type **ftype exefile**.

You'll see the file association is set to

```
exefile="%1" %*
```

Thus, when you run an .exe file, Windows knows the first value is the command that you want to run and anything else provided are parameters to pass along.

> *TIP* File associations and types are maintained in the Windows Registry and can be set using the ASSOC and FTYPE commands respectively. To create the file association, type **assoc** followed by the extension setting, such as **assoc .pl=perlfile**. To create the file type, set the file type mapping, including how to use parameters supplied with the command name, such as **perlfile=C:\Perl\Bin\Perl.exe "%1" %***. To learn more about setting file associations and types, refer to the documentation for these two commands in Help And Support Center.

Redirecting Standard Input, Output, and Error

By default, commands take input from the parameters specified when they are called by the command shell and then send their output, including errors, to the standard console window. Sometimes, though, you'll want to take input from another source or send output to a file or other output device such as a printer. You may also want to redirect errors to a file rather than the console window. You can perform these and other redirection tasks using the techniques introduced in Table 2-2 and discussed in the sections that follow.

TABLE 2-2 Redirection Techniques for Input, Output, and Errors

REDIRECTION TECHNIQUE	DESCRIPTION
command1 > command2	Sends the output of the first command to be the input of the second command.
command < [path]filename	Takes command input from the specified file path.
command > [path]filename	Sends output to the named file, creating the file if necessary or overwriting it if it already exists.
command >> [path]filename	Appends output to the named file if it exists or creates the file and then writes to it.
command < [path]filename > [path]filename	Gets command input from the specified file and then sends command output to the named file.
command < [path]filename >> [path]filename	Gets command input from the specified file and then appends command output to the named file.
command 2> [path]filename	Creates the named file and sends any error output to it. If the file exists, it is overwritten.
command 2>&1	Sends error output to the same destination as standard output.

Redirecting Standard Output to Other Commands

Most commands generate output that can be redirected to another command as input. To do this, you use a technique called *piping*, whereby the output of a command is sent as the input of the next command. Following this, you can see the general syntax for piping is

```
Command1 | Command2
```

where the pipe redirects the output of Command1 to the input of Command2. But you can also redirect output more than once, such as

```
Command1 | Command2 | Command3
```

The two most common commands that are piped include FIND and MORE. The FIND command searches for strings in files or in text passed to the command as input and then lists the text of matching lines as output. For example, you could obtain a list of all .txt files in the current directory by typing the following command:

```
dir | find /I ".txt"
```

The MORE command accepts output from other commands as input and then breaks this output into sections which can be viewed one console page at a time. For example, you could page through a log file called Dailylog.txt using the following command:

```
type c:\working\logs\dailylog.txt | more
```

Type **find /? or more /?** at the command line to get a complete list of the syntax for these commands.

Redirecting I/O to and from Files

Another command redirection technique is to get input from a file using the input redirection symbol (<). For example, the following command sorts the contents of the Usernames.txt file and displays the results to the command line:

```
sort < usernames.txt
```

Just as you can read input from a file, you can also send output to a file. To do this, you can use > to create or overwrite a named file, or >> to create or append data to a named file. For example, if you want to write the current network status to a file, you could use the following command:

```
netstat -a > netstatus.txt
```

Unfortunately, if there is an existing file in the current directory with the same file name, this command overwrites the file and creates a new one. If you want to append this information to an existing file rather than overwrite an existing file, change the command text to read as follows:

```
netstat -a >> netstatus.txt
```

The input and output redirection techniques can be combined as well. You could, for example, obtain command input from a file and then redirect command-output to another file. In this example, a list of user names is obtained from a file and sorted, and then the sorted name list is written to a new file:

```
sort < usernames.txt > usernames-alphasort.txt
```

Redirecting Standard Error

By default, errors from commands are written as output on the command line. If you are running unattended batch scripts or utilities, however, you may want to redirect standard error to a file so that errors are tracked. One way to redirect standard error is to tell the command line that errors should go to the same destination as standard output. To do this, type the **2>&1** redirection symbol as shown in this example:

```
chkdsk /r > diskerrors.txt 2>&1
```

Here, you send standard output and standard error to a file called Diskerrors.txt. If you want to track only errors, you can redirect only the standard error. In this example, standard output is displayed at the command line and standard error is sent to the file Diskerrors.txt:

```
chkdsk /r 2> diskerrors.txt
```

Chaining and Grouping Commands

In previous sections, I discussed redirection techniques that included piping commands. You may have wondered if there were other ways to execute a series of commands. There are. You can chain commands and execute them in sequence, and you can execute commands conditionally based on the success or failure of previous commands. You can also group sets of commands that you want to execute conditionally.

You'll learn more about these techniques in the sections that follow. Before you proceed however, take note of Table 2-3, which provides a quick reference for the basic syntax to use when chaining or grouping commands. Keep in mind that the syntax provided is not intended to be all-inclusive. The chaining syntax can be extended for additional commands to be conditionally executed. The syntax for grouping may vary, depending on the actual situation.

TABLE 2-3 Quick Reference for Chaining and Grouping Commands

SYMBOL	**SYNTAX**	**DESCRIPTION**
&	Command1 & Command2	Execute Command1 and then execute Command2.
&&	Command1 && Command2	Execute Command2 if Command1 is completed successfully.
||	Command1 || Command2	Execute Command2 only when Command1 doesn't complete successfully.
( )	(Command1 & Command2) && (Command3)	Use parentheses to group sets of commands for conditional execution based on success.

(Command1 & Command2) || (Command3)

Use parentheses to group sets of commands for conditional execution based on failure.

Using Chains of Commands

Sometimes, to be more efficient, you'll want to execute commands in a specific sequence. For example, you may want to change to a particular directory and then obtain a directory listing, sorted by date. Using chaining, you can perform both tasks by entering this one line of command text:

```
cd c:\working\docs & dir /O:d
```

In scripts, you'll often need to chain commands such as this to be certain the commands are carried out exactly as you expect. Still, it makes more sense to chain commands when the execution of later commands depends upon whether previous commands succeeded or failed. In this example, a log file is moved only if it exists:

```
dir c:\working\logs\current.log && move current.log d:\history\logs
```

Why would you want to do this? Well, one reason would be so that an error isn't generated as output of a script.

You may also want to perform a task only if a preceding command failed. For example, if you are using a script to distribute files to a group of workstations, some of which have a C:\Working\Data folder and some of which have a C:\Data folder, you could copy sets of files to either folder, regardless of the workstation configuration, using the following commands:

```
cd C:\working\data || cd C:\data
xcopy n:\docs\*.*
```

Grouping Command Sequences

When you combine multiple commands, you may need a way to group commands to prevent conflicts or to ensure that an exact order is followed. You group commands using a set of parentheses. To understand why grouping may be needed, consider the following example. Here, you want to write the host name, IP configuration, and network status to a file, so you use this statement:

```
hostname & ipconfig & netstat -a > current_config.log
```

When you examine the log file, however, you find that it contains only the network status. The reason for this is that the command line executes the commands in the following sequence:

1. hostname
2. ipconfig
3. netstat -a > current_config.log

Because the commands are executed in sequence, the system host name and IP configuration are written to the command line, and only the network status is written to the log file. To write the output of all the commands to the file, you would need to group the commands as follows:

```
(hostname & ipconfig & netstat -a) > current_config.log
```

Here, the output of all three commands is collected and then redirected to the log file. You can also use grouping with conditional success and failure. In the following example, both Command1 and Command2 must succeed for Command3 to execute:

```
(cd C:\working\data & xcopy n:\docs\*.*) && (hostname >
n:\runninglog.txt)
```

In the next chapter, you'll see how command grouping is used with if and if...else constructs.

Chapter 3. Command-Line Scripting Essentials

In a world dominated by whiz-bang graphical user interfaces, you may wonder what command-line scripting has to offer that Windows and point-and-click dialog boxes don't. Well, to be honest, the plain appearance of command-line scripting has more to offer than most people realize, especially considering that most people regard command-line scripts as glorified batch files—the kind you used on computers with 8088 processors and MS-DOS. Today's command-line scripting environment is an extensive programming environment, which includes the following:

- Variables
- Arithmetic expressions
- Conditional statements
- Control flow statements
- Procedures

You can use these programming elements to automate repetitive tasks, perform complex operations while you're away from the computer, find resources that others may have misplaced, and perform many other time-saving activities that you would normally have to type in at the keyboard. Command-line scripts not only have complete access to the command line, they can also call any utilities that have command-line extensions.

Creating Command-Line Scripts

Command-line scripts are text files containing the commands you want to execute. These are the same commands you would normally type into the Windows command shell. However, rather than enter the commands each time you want to use them, you create a script to store the commands for easy execution.

Because scripts contain standard text characters, you can create and edit scripts using a standard text editor, such as Notepad. When you enter commands, be sure to place each command or group of commands that should be executed together on a new line. This ensures proper execution of the commands. When you have finished creating a command-line script, save the script file using the .bat or .cmd extension. Both extensions work with command-line scripts in the same way. For example, if you wanted to create a script to display the system name, the Windows version, and the IP configuration, you could enter these three commands into a file called SysInfo.bat or SysInfo.cmd:

```
hostname
ver
ipconfig -all
```

When you save the script, you can execute it as if it were a Windows utility: Simply type the name of the script in a command shell and press Enter. When you do this, the command shell reads the script file and executes its commands one by one. It stops executing the script when it reaches the end of the file or reads an EXIT command. For the example script, the command line would display output similar to Listing 3-1.

LISTING 3-1 Output of Sample Script

```
C:\>hostname
mailer1

C:\>ver
Microsoft Windows [Version 6.3.9600]

C:\>ipconfig -all
Windows IP Configuration
   Host Name . . . . . . . . . . . . : mailer1
   Primary Dns Suffix  . . . . . . . : imaginedlands.com
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : imaginedlands.com

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) PRO/100 VE Network
      Connection
   Physical Address. . . . . . . . . : X0-EF-D7-AB-E2-1E
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::2ca3:3d2e:3d46:fe99%9
                                       (Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.10.50
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.10.1
   DNS Servers . . . . . . . . . . . : ::1
                                     192.168.10.155
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

If you examine the listing, you'll see that the command prompt and the actual commands are displayed as well as the output of the commands themselves. The reason for this is that the command shell does some extra work behind the scenes while executing scripts in the default processing mode. First, the command shell displays the command prompt. Next, it reads a line from the script, displays it, and then interprets it. If the command shell reaches the end of the file or reads an EXIT command, execution stops. Otherwise, the command shell starts this process all over again by displaying the prompt and preparing to read the next line in the script.

Although the default processing mode with command echoing turned on can be useful for troubleshooting problems in scripts, you probably don't want to use this display mode with scripts you'll use regularly. Fortunately, you can change the default behavior by turning command echo off, as I'll show you in the section titled "Managing Text Display and Command Echoing" later in the chapter.

Common Statements and Commands for Scripts

So far in this book, I've discussed commands but haven't really introduced what a statement is. While these terms are often used interchangeably, the term *statement* technically refers to the keyword for a command, such as the *rem* statement—but it can also refer to a line of code that includes all the command text on that line. In some programming languages, such as Java, each statement must be terminated with a specific character. With Java, the terminator is a semicolon. The command line doesn't look for a specific terminator other than the end of the line, which is assumed when the command interpreter reads any of the following:

- Line break (such as when you press Shift+Enter)
- Carriage return and line break (such as when you press Enter)
- End-of-file marker

Now that we've discussed how to create scripts, let's look at common statements and commands you'll use in scripts, including the following:

- **Cls** Clears the console window and resets the screen buffer
- **Rem** Creates comments in scripts
- **Echo** Displays messages at the command line and turns command echoing on or off
- **@** Controls command echo on a line-by-line basis
- **Title** Sets the title for the command-shell window
- **Color** Sets the text and background colors used in the command-shell window

Clearing the Command-Shell Window

Clearing the command-shell window before writing script output is usually a good idea. You clear the command-shell window using the CLS command. Why not try it? At the command line, type **cls** and press Enter. The console window clears and the cursor is positioned in the top left corner of the window, immediately following the command prompt. All other text in the screen buffer is cleared.

You could add the CLS command to the sample script listed previously, as shown in this example:

```
cls
hostname
ver
ipconfig -all
```

Adding Comments to Scripts

You use the *rem* statement to add comments to your scripts. Every script you create should have comments that include the following details:

- When the script was created and last modified
- Who created the script
- What the script is used for
- How to contact the script creator
- Whether and where script output is stored

Not only are the answers to the questions of who, what, how, where, and when important for ensuring that the scripts you create can be used by other administrators, they can also help you remember what a particular script does, especially if weeks or months have passed since you last worked with the script. An example of a script that uses comments to answer these questions is shown as Listing 3-2.

LISTING 3-2 Updated Sample Script with Comments

```
rem ************************
rem Script: SysInfo.bat
rem Creation Date: 2/28/2015
rem Last Modified: 3/15/2015
rem Author: William R. Stanek
rem E-mail: williamstanek@aol.com
rem ************************
rem Description: Displays system configuration information
rem              including system name, IP configuration
rem              and Windows version.
rem ************************
rem Files: Stores output in c:\data\current-sys.txt.
rem ************************

hostname > c:\data\current-sys.txt
ver >> c:\data\current-sys.txt
ipconfig -all >> c:\data\current-sys.txt
```

Later in this chapter, in the section titled “Passing Arguments to Scripts,” I’ll show you how to use your comments as automated help documentation. Before we get to that, however, keep in mind that you can also use *rem* statements to

- Insert explanatory text within scripts, such as documentation on how a procedure works.
- Prevent a command from executing. On the command line, add **rem** before the command to comment it out.
- Hide part of a line from interpretation. Add **rem** within a line to block interpretation of everything that follows the *rem* statement.

Managing Text Display and Command Echoing

The ECHO command has two purposes. You use the ECHO command to write text to the output, which can be the command shell or a text file. You also use the ECHO command to turn command echoing on or off. Normally, when you execute commands in a script, the commands as well as the resulting output of the command are displayed in the console window. This is called *command echoing*.

To use the ECHO command to display text, enter echo followed by the text to display, such as

```
echo The system host name is:
hostname
```

To use ECHO to control command echoing, type **echo off** or **echo on** as appropriate, such as

```
echo off
echo The system host name is:
hostname
```

Use output redirection to send output to a file rather than the command shell, as follows:

```
echo off
echo The system host name is: > current.txt
hostname >> current.txt
```

To experiment with suppressing command echoing, start a command shell, type **echo off**, and then enter other commands. You'll find that the command prompt is no longer displayed. Instead, you see only what you type into the console window and the resulting output from the commands you've entered. In scripts, the ECHO OFF command turns off command echoing as well as the command prompt. By adding the command ECHO OFF to your scripts, you keep the command-shell window or the output file from getting cluttered with commands when all you care about is the output from those commands.

> *TIP* By the way, if you want to determine whether command echoing is enabled or disabled, type the ECHO command by itself. Give it a try. If command echoing is on, you'll see the message Echo Is On. Otherwise, you'll see the message Echo Is Off. Experiment with the ECHO OFF command in your scripts and you may detect a bit of a problem here. If the ECHO OFF command turns off command echoing, how do you prevent the ECHO OFF command itself from echoing? Don't worry; that's discussed in the next section.

> *REAL WORLD* Other command-line programmers frequently ask me how to get a blank line to echo in the command shell. You might think that putting the ECHO command on a line by itself would do the job, but it doesn't. Typing **echo** on a line by itself displays the status of command echoing, as mentioned in the previous tip. Typing **echo** followed by a space doesn't work either, because the Windows command line treats spaces (in this situation) as meaningless, and you get the same results as typing **echo** followed by nothing at all. To get ECHO to display a blank line, you must enter **echo** and a period **(echo.)**. The period is part of the command text and there is no space between the period and the ECHO command.

Fine-Tuning Command Echo with @

The @ command prevents commands from echoing to the output on a line-by-line basis; you can think of it as a line-specific *echo off* statement.

You could use @ to turn off command echoing like this:

```
@echo The system host name is:
@hostname
```

Using @, the output that shows the command prompt and commands like this:

```
C:\>echo The system host name is:
The system host name is:

C:\>hostname
mailer1
```

becomes

```
The system host name is:
mailer1
```

But the real value of @ is that it allows you to tell the command shell not to display the command prompt or ECHO OFF command, and thereby ensures that the only output of your script is the output of the commands you enter. Here is an example of a script that uses @ to hide the ECHO OFF command so that it isn’t displayed in the output:

```
@echo off
echo The system host name is:
hostname
```

The output from this script is

```
The system host name is:
mailer1
```

> *TIP* I recommend using @echo off at the beginning of all your command-line scripts. By the way, if you start a command shell and type **@echo off**, you can turn off the display of the command prompt as well.

Setting the Console Window Title and Colors

If you’re going to take the time to write a script, you might as well add a

If you're going to take the time to write a script, you might as well add a few special features to jazz it up. Some of the basic techniques that I've already discussed are using the ECHO OFF command and clearing the console window before you write output. You may also want to set a title for the window or change the colors the window uses.

The title bar for the command shell is located at the top of the console window. Normally, this title bar displays Command Prompt or the file path to the command shell. You can customize the title using the TITLE command. This command works much like the ECHO command in that it displays whatever text follows it on the console's title bar. For example, if you wanted to set the title of the current console to System Information, you would enter the following at the command line:

```
title System Information
```

Not only can you use the TITLE command to show the name of the script that is running, but you can also use TITLE to show the progress of the script as it executes, such as

```
rem add blocks of work commands
title Gathering Information

rem add blocks of logging commands
title Logging System Information
```

By default, the console window displays white text on a black background. As you learned in Chapter 1, "Overview of the Windows Command Line," you can modify this behavior using the Colors tab of the Command Prompt Properties dialog box. You can also set console colors by using the COLOR command. You do this by passing the command a two-digit hexadecimal code. The first digit corresponds to the background color and the second digit corresponds to the text color, as the following example, which sets the text to blue and the background color to green:

```
color 21
```

The color codes you can use with the COLOR command are shown in Table 3-1. Keep in mind that you can't set the text and background colors to the same value. If you try to do this, the color doesn't change.

to the same value. If you try to do this, the color doesn't change. Additionally, you can restore the default colors at any time by using the COLOR command without any arguments, like so:

```
color
```

TABLE 3-1 Color Codes for the Command-Shell Window

CODE	COLOR	CODE	COLOR
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	A	Light Green
3	Aqua	B	Light Aqua
4	Red		

C	Light Red
5	Purple
D	Light Purple
6	Yellow
E	Light Yellow
7	White
F	Bright White

Passing Arguments to Scripts

As with most command-line utilities, you can pass arguments to scripts when they are started. You use arguments to set special parameters in a script or to pass along information needed by the script. Each argument should follow the script name and be separated by a space (and enclosed in quotation marks if necessary). In the following example, a script named *check-sys* is passed the arguments *mailer1* and *full*:

```
check-sys mailer1 full
```

Each value passed along to a script can be examined using formal parameters. The script name itself is represented by the parameter *%0*. The parameter *%1* represents the first argument passed in to the script, *%2* the second, and so on until *%9* for the ninth argument. For example, if you create a script called check-sys and then use the following command to call the script:

```
check-sys mailer1 full actual
```

you would find that the related parameter values are

- %0 — check-sys
- %1 — mailer1
- %2 — full
- %3 — actual

You access arguments in scripts using the parameter name: *%0* for the script name, *%1* for the first script parameter, and so on. For example, if you wanted to display the script name and the first argument passed to the script, you could enter the following:

```
echo %0
echo %1
```

If you pass in more than nine parameters, the additional parameters are not lost. Instead, they are stored in a special parameter: *%** (percent + asterisk). The *%** parameter represents all arguments passed to the script and you can use the SHIFT command to examine additional parameters. If you call SHIFT without arguments, the script parameters are shifted by 1. This means the related value for %0 is discarded and replaced by the related value for *%1*, and the related value for %2 becomes the related value for *%1*, and so on. You can also specify where shifting begins so that you can retain previous parameters if necessary. For example, if you use the following command, *%4* becomes *%3*, *%5* becomes *%4*, and so on. But *%0*, *%1*, and *%2* are unaffected:

```
shift /3
```

Getting Acquainted with Variables

In command-line scripting, what we commonly call variables are more properly called *environment variables*. Environment variables can come from many sources. Some variables are built into the operating system or derived from the system hardware during startup. These variables, called *built-in system variables*, are available to all Windows processes regardless of whether anyone is logged on interactively. System variables can also come from the Windows Registry. Other variables are set during logon and are called *built-in user variables*. The built-in user variables available are the same, no matter who is logged on to the computer. As you might expect, they are valid only during an actual logon session—that is, when a user is logged on.

You can see a listing of all the variables known in the current instance of the command shell by typing **set** at the prompt. In addition to the normal system and user variables, you can create variables whenever Windows is running, which is exactly what you'll do when you program in the command shell. You define variables for the current instance of the command shell using the SET command and the following syntax:

```
set variable_name=variable_value
```

such as

```
set working=C:\Work\Data
set value=5
set string="Hello World"
```

Some variables, including system and user environment variables, have special meaning in the command shell. These variables include *path*, *computername*, *homedrive*, and many other important environment variables. One environment variable that you should learn more about is *errorlevel*, which tracks the exit code of the most recently used command. If the command executes normally, the error level is zero (0). If an error occurs while executing the command, the error level is set to an appropriate nonzero value. Error values include

- **1** Indicates a general error
- **2** Indicates an execution error, meaning the command failed to execute properly

- **–2** Indicates a math error, such as when you create a number that is too large for the command shell to handle

You can work with the *errorlevel* variable in several ways. You can check for a specific error condition, such as

```
if "%ERRORLEVEL%"=="2" echo "An error occurred!"
```

Or, you can use the following special syntax and check for a condition equal to or greater than the specified exit code:

```
if errorlevel 2 echo "An error occurred!"
```

> *NOTE* You'll see more on *errorlevel* and *if* statements later in the chapter in the sections titled "Substituting Variable Values" and "Command-Line Selection Statements."

When you are finished working with variables, it's good form to dispose of them. You do this to free memory used by the variable and prevent problems or unexpected results if you accidentally refer to the variable in the future. To clear out a variable, you simply set the variable equal to nothing, such as

```
set working=
```

Now the variable is cleared out of memory and is no longer available.

Using Variables in Scripts

In scripts, you'll use variables to store values as you perform various types of operations. Unlike most programming languages, you cannot declare a variable in a command-line script without simultaneously assigning it a value. This makes a certain amount of sense because from a practical point of view, there's no reason to have a variable that contains nothing. The sections that follow discuss key concepts for working with variables, including

- Variable names
- Variable values
- Variable substitution
- Variable scope

Naming Variables

The command shell tracks variable names in the case you use but doesn’t care about the case when you are working with the variable. This means variable names aren’t case-sensitive but are case-aware. Beyond this, very few restrictions apply to variable names and you can use just about any combination of letters, numbers, and characters to form the variable name. In fact, all the following variable names are technically valid:

```
2six
85
!
?
```

Why in the world you’d want to use such horrendous variable names, however, is beyond me. With that said, how should you name your variables? Well, the most important rule to keep in mind is that variable names should be descriptive. Use names such as

```
Systemname
CurrentStats
mergetotal
Net_Address
```

These descriptive variable names are helpful when you or someone else needs to modify the script. And notice that you have many ways to create multiple-word variable names. Although you are free to use whatever style you like, most programmers format multiword variable names with a lowercase initial letter on the first word and uppercase initial letter on each subsequent word. Why? Simply because this is a standard naming convention. Following this convention, the variable names listed previously would be

```
systemName
```

```
currentStats
mergeTotal
netAddress
```

> *NOTE* Keep in mind that the command shell doesn't care about the case. Variable names are case-aware but they're not case-sensitive. This means that you could refer to the *systemName* variable as *SYSTEMNAME*, *systemname*, or even *sYStemNAMe*.

Setting Variable Values

As discussed previously, you define new variables using the following syntax, where *variable_name* is the variable name and *variable_value* is its related value:

```
set variable_name=variable_value
```

Spaces are valid in both names and values. So only use spaces around the equal sign (=) if you want the name and/or the value to include these spaces.

Unlike many programming languages, the command shell doesn't differentiate between various data types. All variables are stored as character strings. This is true even when you set the variable value to a number. Thus, the following values are stored as strings:

```
Current status:
311
"Error!"
12.75
```

using commands such as:

```
set varA=Current status:
set varB=311
set varC="Error!"
set varD=12.75
```

Don't forget that some characters are reserved in the command line, including @ < > & | ^. Before you use these characters, you must escape them with the caret symbol (^)—no matter where they occur in the variable value. (This is discussed in Chapter 2, "Getting the Most from the Command Line.") For example, to set these literal string values:

```
2 & 3 = 5
2^3
```

you must set the variable value as follows:

```
2 ^& 3 = 5
2^^3
```

using statements such as

```
set example1=2 ^& 3 = 5
set example2=2^^3
```

> *NOTE* An odd thing happens if you try to echo the example values. Instead of the equations you expect, you get either an error or an odd value. What is happening here is that when you echo the value, the special characters are reparsed. If you want to set a variable to a value that includes a special character and also be able to display this value to users, you must use three escape codes, meaning that you would use set *example1=2 ^^^& 3 = 5 or set example2=2^^^^3*. This is necessary because the value is double parsed (once when the value is set and once when the value is displayed).

Substituting Variable Values

Variables wouldn't be very useful if the only way you could access them was with the SET command. Fortunately, you can access variable values in other ways. One of these ways is to use variable substitution to compare a variable name with its actual value. You saw this type of substitution at work in the following line from a previous example in this chapter:

```
if "%ERRORLEVEL%"=="2" echo "An error occurred!"
```

Here, you are determining whether the value of the errorlevel environment variable is equal to 2 and, if it is, you display text stating that an error occurred. The percent signs surrounding the variable name tell an error occurred. The percent signs surrounding the variable name tell the command shell you are referencing a variable. Without these percent signs, Windows would perform a literal comparison of "ERRORLEVEL" and "2". Note also the use of quotation marks in the example. The quotation marks ensure an exact comparison of string values.

Another way to use substitution is to replace a variable name with its actual value. For example, you might want to create a script that can be run on different computers, so rather than hard-coding the path to the system root directory as C:\Windows, you could use the environment variable *systemroot*, which references the system root of the particular computer being accessed. With this in mind, you use the following line of code in your script:

```
cd %SYSTEMROOT%\System32
```

instead of this line of code:

```
cd C:\Windows\System32
```

You can also use variable substitution when you are assigning variable values, such as

```
systemPath=%SystemRoot%\System32
```

Variable substitution can be quite powerful. Consider the code snippet shown in Listing 3-3.

LISTING 3-3 Sample Script Header

```
@echo off
@if not "%OS%"=="Windows_NT" goto :EXIT
@if "%1"=="" (set INFO=echo && set SEXIT=1) else (set INFO=rem && set
SEXIT=0)

%INFO% ************************
%INFO% Script: SystemInfo.bat
%INFO% Creation Date: 2/28/2015
%INFO% Last Modified: 3/15/2015
%INFO% Author: William R. Stanek
%INFO% E-mail: williamstanek@aol.com
%INFO% ************************
%INFO% Description: Displays system configuration information
```

```
%INFO%              including system name, IP configuration
%INFO%              and Windows version.
%INFO% ************************
%INFO% Files: Stores output in c:\current-sys.txt.
%INFO% ************************

@if "%SEXIT%"=="1" goto :EXIT

@title "Configure Scheduling..."
cls
color 07
```

Listing 3-3 is a standard header that I use in some of my scripts. The first *if* statement checks to see what operating system is running. If it is Windows, the script continues execution. Otherwise a *goto* subroutine is called. The second *if* statement checks the value of the first argument passed in to the script. If the script is called with no arguments, instances of *%INFO%* are replaced with *echo*, which writes the script documentation to the output. If the script is called with one or more arguments, instances of *%INFO%* are replaced with *rem* to designate that the associated lines are comments.

> *NOTE* Don't worry if you don't understand the example completely. You'll learn all about conditional execution and subroutines in the sections titled, "Command-Line Selection Statements" and "Creating Subroutines and Procedures" later in the chapter.

Localizing Variable Scope

Changes you make to variables in the command shell using set are localized, meaning that they apply only to the current command shell instance or to command shells started within the current command shell (nested command shells) and are not available to other system processes. Further, when you exit the command shell in which variables were created, the variables no longer exist.

Sometimes you may want to limit the scope of variables even further than their current command-shell process. To do this, you can create a local scope within a script that ensures that any variable changes are localized to that specific area within the script. Later, you can end the local scope and restore the environment to its original settings.

You can mark the start of a local scope within a script using the SETLOCAL command and then end the local scope with an ENDLOCAL command. Several events take place when you use these commands. The call to SETLOCAL creates a snapshot of the environment. Any changes you make within the scope are then localized and discarded when you call ENDLOCAL. The following example uses SETLOCAL and ENDLOCAL:

```
@echo off
set sysCount=0
set deviceCount=0

rem Start localization
setlocal
set sysCount=5
set deviceCount=5
echo Local count: %sysCount% system edits ^& %deviceCount% device checks
endlocal

echo Count: %sysCount% system edits ^& %deviceCount% device checks
```

The output of the script is

```
Local count: 5 system edits & 5 device checks
Count: 0 system edits & 0 device checks
```

As you can see, local scopes behave much like nested command shells. As with the nested command shells, you can nest several layers of localization. And though each layer inherits the environment settings of its parent, any changes in the nested layer are not reflected in the parent environment.

Using Mathematical Expressions

At times, you'll want to perform some kind of mathematical operation in your scripts and assign the results to a variable. As with most programming languages, the command shell allows you to write mathematical expressions using a variety of operators, including the following:

- Arithmetic operators to perform standard mathematical operations (such as addition, subtraction, multiplication, and division)
- Assignment operators that combine an assignment operation (symbolized by the equal sign) with an arithmetic operation
- Comparison operators that compare values and are usually used with *if* statements
- Bitwise operators that allow you to manipulate the sequences of binary values

Math operations are performed using SET with the */A* (arithmetic) parameter, such as

```
set /a theTotal=18+2
set /a theTotal=18*2
set a theTotal=182
```

All mathematical expressions are evaluated using 32-bit signed integer arithmetic. This allows for values –2^32 to 2^32-1. If you exceed this range, you'll get an arithmetic error (code –2) instead of the intended value.

The most commonly used operators are those for arithmetic, assignment, and comparison. Arithmetic and assignment operators are discussed in the sections that follow. Comparison operators are discussed in the section titled "Making Comparisons in If Statements" later in this chapter. Pay particular attention to the additional discussions on operator precedence and simulating exponents.

Working with Arithmetic and Assignment Operators

You use arithmetic operators to perform basic math operations on numerical values. These values can be expressed literally as a number, such as 5, or as a variable that contains the value you want to work with, such as *%TOTAL%*.

Table 3-2 summarizes the available arithmetic and assignment operators.

Most of the arithmetic operators are fairly straightforward. You use * in multiplication, / in division, + in addition, and – in subtraction. You use the equal sign (=) to assign values to variables. You use % (modulus) to obtain the remainder from division. For example, if you divide 8 into 60, the answer is 7 Remainder 4; the value 4 is what the result would be if you use the modulus operator.

Examples of working with arithmetic operators follow:

```
set /a theCount=5+3
set /a theCount=%nServers% + %nWstations%
set /a theCount=%nServers% - 1
```

> *TIP* Earlier, I stated that everything stored in a variable is a string, and that remains true. However, the command shell can detect when a string contains only numerals, and this is what allows you to use variables in expressions. The key detail to remember is to use the proper syntax for substitution, which is *%variableName%.*

TABLE 3-2 Arithmetic and Assignment Operators

ARITHMETIC OPERATORS	ASSIGNMENT OPERATORS
+ (Addition)	+= (Increment, or add and assign)
- (Subtraction)	-= (Decrement, or subtract and assign)
* (Multiplication)	*= (Scale up, or multiply and assign)
/ (Division)	/= (Scale down, or divide and assign)
% (Modulus)	%= (Modulus and assign)

You use assignment operators to increment, decrement, scale up, or scale down. These operators combine arithmetic and assignment operation functions. For example, the += operator is used to increment a value and combines the effects of the + operator and the = operator. Thus, the following two expressions are equivalent and yield identical results when entered at the command line:

```
set /a total=total+1
set /a total+=1
```

Understanding Operator Precedence

One thing you should understand when working with mathematic operations is *operator precedence*. Operator precedence determines what happens when the command shell must evaluate an expression that involves more than one operator. For example:

```
set /a total=8+3*4
```

If evaluated from left to right, this expression equals 44 (8+3=11, 11*4=44). But as in standard mathematics, that's not how the command line evaluates the expression. Instead, the command shell evaluates the expression as 20 (3*4=12, 8+12=20) because the precedence of operations is the following:

1. Modulus
2. Multiplication and division
3. Addition and subtraction

> *NOTE* When an expression contains multiple operations at the same precedence level, these operations are performed from left to right. Hence, set /a total=10-4+2 equals 8 (10-4=6, 6+2=8).

However, as with standard mathematics, you can use parenthetical grouping to ensure that numbers are processed in a certain way. This means you can use the expression

```
set /a total=(8+3)*4
```

to ensure that the command-line interprets the expression as (8+3=11, 11*4=44).

Simulating Exponents

Although you can perform many mathematical operations at the command line, you have no way to raise values to exponents. You can, however, perform these operations manually. For example, the easiest way to get a value for 2^3 is to enter

```
set /a total=2*2*2
```

The result is 8. Similarly, you can get a value for 10^5 by entering

```
set /a total=10*10*10*10*10
```

The result is 100,000.

Command-Line Selection Statements

Now that you know how to work with variables and form expressions, let's look at something more advanced: selection statements used with the command line. When you want to control the flow of execution based upon conditions known only at run time, you'll use

- *if* to execute a statement when a condition is true, such as if the operating system is Windows. Otherwise, the statement is bypassed.
- *if not* to execute a statement when a condition is false, such as if a system doesn't have a C:\Windows directory. Otherwise, the statement is bypassed.
- *if...else* to execute a statement if a condition is matched (true or false) and to otherwise execute the second statement.

Although some of the previous examples in this chapter have used conditional execution, we haven't discussed the syntax for these conditional execution, we haven't discussed the syntax for these statements or the associated comparison operators. If your background doesn't include programming, you probably will be surprised by the power and flexibility of these statements.

Using If

The *if* statement is used for conditional branching. It can be used to route script execution through two different paths. Its basic syntax is

```
if condition (statement1) [else (statement2)]
```

Here each statement can be a single command or multiple commands chained, piped, or grouped within parentheses. The condition is any expression that returns a Boolean value of True or False when evaluated. The *else* clause is optional, meaning you can also use the following syntax:

```
if condition (statement)
```

> *TIP* Technically, parentheses aren't required, but using them is a good idea, especially if the condition includes an echo statement or a command with parameters. If you don't use parentheses in these instances, everything that follows the statement on the current line will be interpreted as part of the statement, which usually results in an error.

The *if* statement works like this: If the *condition* is true, *statement1* is executed. Otherwise, *statement2* is executed (if the else clause is provided). In no case will both the *if* and the *else* clauses be executed. Consider the following example:

```
if "%1"=="1" (echo is one) else (echo is not one)
```

Here, if the first parameter passed to the script is 1, "is one" is written to the output. Otherwise, "is not one" is written to the output.

The command shell expects only one statement after each condition. Typically, the statement is a single command to execute. If you want to execute multiple commands, you'll need to use one of the command execute multiple commands, you'll need to use one of the command piping, chaining, or group techniques, as in this example:

```
if "%1"=="1" (hostname & ver & ipconfig /all) else (netstat -a)
```

Here all three commands between the first set of parentheses will execute if the first parameter value is 1.

Using If Not

When you want to execute a statement only if a condition is false, you can use *if not*. The basic syntax is

```
if not condition (statement1) [else (statement2)]
```

Here the command shell evaluates the *condition*. If it is false, the command shell executes the statement. Otherwise, *statement1* doesn't execute and the command shell proceeds to *statement2*, if present. The *else* clause is optional, meaning you can also use the following syntax:

```
if not condition (statement1)
```

Consider the following example:

```
if not errorlevel 0 (echo An error has occurred!) & (goto :EXIT)
```

Here you check for error conditions other than zero. If no error has occurred (meaning the error level is zero), the command shell continues to the next statement. Otherwise, the command shell writes "An error has occurred!" to the output and exits the script. (You'll learn all about *goto* and subroutines later in the chapter.)

Using If Defined and If Not Defined

The final types of *if* statements you can use are *if defined* and *if not defined*. These statements are designed to help you check for the existence of variables, and their respective syntaxes are

```
if defined variable statement
```

and

`if not defined` variable statement

Both statements are useful in your shell scripts. In the first case, you execute a command if the specified variable exists. In the second case, you execute a command if the specified variable does not exist. Consider the following example:

```
if defined numServers (echo Servers: %numServers%)
```

Here, if the *numServers* variable is defined, the script writes output. Otherwise, the script continues to the next statement.

Nesting Ifs

A nested *if* is an *if* statement within an *if* statement. Nested *ifs* are very common in programming, and command-shell programming is no exception. When you nest *if* statements, pay attention to the following points:

- Use parentheses to define blocks of code and the @ symbol to designate the start of the nested *if* statement.
- Remember that an else statement always refers to the nearest *if* statement that is within the same block as the *else* statement and that is not already associated with another *else* statement.

Here is an example:

```
if "%1"=="1" (
@if "%2"=="2" (hostname & ver) else (ver)) else (hostname & ver &
netstat -a)
```

The first *else* statement is associated with if "%2"=="2". The final else statement is associated with if "%1"=="1".

Making Comparisons in If Statements

Frequently, the expression used to control *if* statements will involve comparison operators, as shown in previous examples. The most basic type of string comparison is when you compare two strings using the equality operator (=), such as

```
if stringA==stringB statement
```

Here, you are performing a literal comparison of the strings; if they are exactly identical, the command statement is executed. This syntax works for literal strings but is not ideal for scripts. Parameters and arguments may contain spaces or there may be no value at all for a variable. In this case, you may get an error if you perform literal comparisons. Instead, use double quotation marks to perform a string comparison and prevent most errors, such as

```
if "%varA%"=="%varB%" statement
```

or

```
if "%varA%"=="string" statement
```

String comparisons are always case-sensitive unless you specify otherwise with the /i switch. The /i switch tells the command shell to ignore the case in the comparison, and you can use it as follows:

```
if /I "%1"=="a" (echo A) else (echo is not A)
```

To perform more advanced equality tests, you'll need to use the comparison operators shown in Table 3-3. These operators are used in place of the standard equality operator, such as

```
if "%varA%" equ "%varB" (echo The values match!)
```

TABLE 3-3 Using Comparison Operators

OPERATOR	DESCRIPTION
equ	Checks for equality and evaluates to true if the values are equal

neq

Checks for inequality and evaluates to true if the values are not equal

lss

Checks for less-than condition and evaluates to true if *value1* is less than *value2*

leq

Checks for less-than or equal-to condition and evaluates to true if *value1* is less than or equal to *value2*

gtr

Checks for greater-than condition and evaluates to true if *value1* is greater than *value2*

geq

Checks for greater-than or equal-to condition, and evaluates to true if *value1* is greater than or equal to *value2*

Command Line Iteration Statements

When you want to execute a command or a series of commands repeatedly, you'll use the *for* statement. The *for* statement is a powerful construct, and before you skip this section because you think you know how the *for* statement works, think again. The *for* statement is designed specifically to work with the command-shell environment and is very different from any other *for* statement you may have worked with in other programming languages. Unlike most other *for* statements, the one in the command line is designed to help you iterate through groups of files and directories, and to parse text files, strings, and command output on a line-by-line basis.

Iteration Essentials

The command shell has several different forms of *for* statements. Still, the basic form of all *for* statements is

```
for iterator do (statement)
```

Here the iterator is used to control the execution of the *for* loop. For each step or element in the iterator, the specified *statement* is executed. The *statement* can be a single command or multiple commands chained, piped, or grouped within parentheses.

The iterator usually consists of an initialization variable and a set of elements to execute against, such as a group of files or a range of values to step through. Initialization variables are essentially placeholders for the values you want to work with. When you work with initialization variables, keep in mind the following:

- Iterator variables only exist within the context of a *for* loop.
- Iterator variable names must be in the range from a to z or A to Z, such as *%%A, %%B*, or *%%C*.
- Iterator variable names are case-sensitive, meaning *%%a* is different from *%%A*.

As Table 3-4 shows, the various structures used with *for* statements have specific purposes and forms. When the *for* statement is initialized, iterator variables, such as *%%B*, are replaced with their actual values. These values come from the element set specified in the *for* statement and could consist of a list of files, a list of directories, a range of values, and so on.

TABLE 3-4 Forms for Iteration

ITERATION PURPOSE	**FORM SYNTAX**
Sets of files	for *%%variable* in (fileSet) do *statement*
Sets of directories	for /D *%%variable* in (directorySet) do *statement*
Files in subdirectories	for /R [*path*] *%%variable* in (fileSet) do *statement*

Stepping through a series of values

for /L *%%variable* in (*stepRange*) do *statement*

Parsing text files, strings, and command output

for /F [“*options*”] *%%variable* in (source) do *statement*

REAL WORLD The forms provided are *for* scripts. You can also use *for* statements interactively at the command line. In this case, use *%variable* instead of *%%variable*. Beyond this, *for* statements within scripts or at the command line are handled in precisely the same way.

Stepping Through a Series of Values

The “traditional” way to use *for* statements is to step through a range of values and perform tasks using these values. You can do this in the command shell. Within batch scripts, the basic syntax of this type of *for* loop is

```
for /l %%variable in (start,step,end) do (statement)
```

At the prompt, the basic syntax of this type of *for* loop is

```
for /l %variable in (start,step,end) do (statement)
```

This type of *for* statement operates as follows. First, the command shell initializes internal *start*, *step*, and *end* variables to the values you’ve specified. Next, it compares the start value with the end value to determine whether the statement should be executed, yielding a true condition if the start value can be incremented or decremented as specified in the step and a false condition otherwise. In the case of a true condition, the command shell executes the statement using the start value and then increments or decrements the start value by the step value specified. Afterward it repeats this process. In the case of a false condition, the command shell exits the for statement, moving on to the next statement in the script.

Consider the following example for a script that counts from 0 to 10 by increments of 2:

```
for /l %%B in (0,2,10) do echo %%B
```

The output is

```
0
2
4
6
8
10
```

You can also use a negative step value to move through a range in decreasing values. You could count from 10 to 0 by twos as follows:

```
for /l %%B in (10,-2,0) do echo %%B
```

The output is

```
10
8
6
4
2
0
```

To modify the example to work at the prompt, change %% to %, such as:

for /l %B in (10,-2,0) do echo %B

Iterating Through Groups of Files

A more powerful way to use for statements in the command shell is to use them to work with files and directories. Within scripts, the *for* statement syntax for working with groups of files is

```
for %%variable in (fileSet) do (statement)
```

At a prompt, the *for* statement syntax for working with groups of files is

```
for %variable in (fileSet) do (statement)
```

Here you use *fileSet* to specify a set of files that you want to work with. A file set can be any of the following:

- Individual files as specified by a filename, such as MyFile.txt
- Groups of files specified with wildcards, such as *.txt
- Multiple files or groups of files with spaces separating file names, such as *.txt *.rtf* .doc

Now that you know the basic rules, working with files is easy. For example, if you want to list all text files in an application directory, you can use the following command in a script:

```
for %%B in (C:\Working\*.txt) do (echo %%B)
```

Here B is the initialization variable, *C:\Working\*.txt* specifies that you want to work with all text files in the C:\Working directory, and the statement to execute is *echo %%B*, which tells the command shell to display the current value of *%%B* each time it iterates through the *for* loop. The result is that a list of the text files in the directory is written to the output.

You could extend this example to examine all .txt, .rtf, and .doc files like this:

```
for %%B in (%AppDir%\*.txt %AppDir%\*.rtf %AppDir%\*.doc) do (echo %%B)
```

You can also use multiple commands using piping, grouping, and chaining techniques, such as

```
for %%B in (%AppDir%\*.txt %AppDir%\*.rtf %AppDir%\*.doc) do (echo %%B &
move C:\Data)
```

Here you list the .txt, .rtf, and .doc files in the location specified by the *AppDir* variable and then move the files to the C:\Data directory.

To modify the example to work at the prompt, change %% to %, such as:

```
for %B in (%AppDir%\*.txt %AppDir%\*.rtf %AppDir%\*.doc) do (echo %B &
move C:\Data)
```

Iterating Through Directories

If you want to work with directories rather than files, you can use the following *for* statement style in scripts:

```
for /d %%variable in (directorySet) do (statement)
```

Or this style at a prompt:

```
for /d %variable in (directorySet) do (statement)
```

Here you use *directorySet* to specify the group of directories you want to work with. Iterating directories works exactly like iterating files, except you specify directory paths rather than file paths. If you wanted to list all the base directories under *%SystemRoot%*, you would do this as follows:

```
for /d %%B in (%SystemRoot%\*) do echo %%B
```

To modify the example to work at the prompt, change %% to %, such as:

```
for /d %B in (%SystemRoot%\*) do echo %B
```

On Windows Server, a partial result list would be similar to

```
C:\Windows\ADFS
C:\Windows\ADWS
C:\Windows\AppCompat
C:\Windows\AppReadiness
```

> *NOTE* Note that the *for /d* loop iterates through the specified directory set but doesn't include subdirectories of those directories. To access subdirectories (and indeed the whole directory tree structure), you use *for /r* loops, which I'll discuss in a moment.

You can specify multiple base directories by separating the directory names with spaces, such as

```
for /d %%B in (%SystemRoot% %SystemRoot%\*) do echo %%B
```

Here you examine the %SystemRoot% directory itself and then the directories immediately below it. So now your list of directories would start with C:\Windows (if this is the system root) and continue with the other directories listed previously.

You can also combine file and directory iteration techniques to perform actions against all files in a directory set, such as

```
for /d %%B in (%APPDATA% %APPDATA%\*) do (
@for %%C in ("%%B\*.txt") do echo %%C)
```

The first for statement returns a list of top-level directories under *%APPDATA%*, which also includes *%APPDATA%* itself. The second *for* statement iterates all .txt files in each of these directories. Note the @ symbol before the second for statement. As with *if* statements, this indicates the second *for* statement is nested and is required to ensure proper execution. The double quotations with the file set ("*%%B*.txt*") ensure that directory and filenames containing spaces are handled properly.

Because you'll often want to work with subdirectories as well as directories, the command shell provides for /r statements. Using for */r* statements, you can examine an entire directory tree from a starting point specified as a path. The syntax is

```
for /r [path] %%variable in (fileSet) do (statement)
```

Here *path* sets the base of the directory tree you want to work with, such as C:\. The path is not required, however, and if the path is omitted, the current working directory is assumed.

Using a *for /r* statement, you could extend the previous example to list all .txt files on the C: drive without needing a double *for* loop, as shown here:

```
for /r C:\ %%B in (*.txt) do echo %%B
```

As you can see, *for /r* statements are simpler and more powerful than double *for* loops. You can even combine */r* and */d* without needing a double loop. In this example, you obtain a listing of all directories and subdirectories under *%SystemRoot%*:

```
for /r %SystemRoot% /d %%B in (*) do echo %%B
```

Parsing File Content and Output

Just as you can work with file and directory names, you can also work with the contents of files and the output of commands. To do this, you'll use the following *for* statement style:

```
for /f ["options"] %%variable in (source) do (statement)
```

Here, *options* sets the text-matching options; *source* specifies where the text comes from, which could be a text file, a string, or command output; and *statement* specifies what commands should be performed on matching text. Each line of text in the source is handled like a record, where fields in the record are delimited by a specific character, such as a tab or a space (which are the default delimiters). Using substitution, the command shell then replaces placeholder variables in the statement with actual values.

Consider the following line of text from a source file:

```
William Stanek Engineering Williams@imaginedlands.com 3408
```

One way of thinking of this line of text is as a record with five fields:

- First Name William
- Last Name Stanek
- **Department** Engineering
- **E-Mail Address** Williams@imaginedlands.com
- Phone Extension 3408

To parse this and other similar lines in the associated file, you could use the following *for* statement:

```
for /f "tokens=1-5" %%A in (current-users.txt) do (
@echo Name: %%A %%B Depart: %%C E-mail: %%D Ext: %%E)
```

Here you specify that you want to work with the first five fields (token fields separated by spaces or tabs by default) and identified by iterator variables, starting with *%%A*, which means the first field is *%%A*, the second *%%B*, and so on. The resulting output would look like this:

```
Name: William Stanek Depart: Engineering E-Mail:
Williams@imaginedlands.com Ext: 3408
```

Table 3-5 shows a complete list of options that you can use. Examples and descriptions of the examples are included.

TABLE 3-5 Options for File Content and Command Output Parsing

OPTION	OPTION DESCRIPTION	EXAMPLE	EXAMPLE DESCRIPTION
eol	Sets the end-of-line comment character. Everything after the end-of-line comment character is considered to be a comment.	"eol=#"	Sets # as the end-of-line comment character.
skip	Sets the number of lines to skip at the beginning of files.	"skip=5"	Tells the command shell to skip lines 1 through 5 in the source file.
delims	Sets delimiters to use for fields. The defaults are space and tab.	"delims=, . :"	

Specifies that commas, periods, and colons are delimiters.

tokens

Sets which token fields from each source line are to be used. You can specify up to 26 tokens provided that you start with a or A as the first iterator variable. By default, only the first token is examined.

"tokens=1, 3"

"tokens=2-5"

First example sets fields to use as 1 and 3. Second example sets fields 2, 3, 4, and 5 as fields to use.

usebackq

Specifies that you can use quotation marks in the source designator: double quotation marks for file names, back quotation marks for command to execute, and single quotation marks for a literal string.

"usebackq"

Enables the option.

To see how additional options can be used, consider the following example:

```
for /f "skip=3 eol=; tokens=3-5" %%C in (current-users.txt) do (
@echo Depart: %%C E-mail: %%D Ext: %%E)
```

Here, three options are used. The *skip* option is used to skip the first three lines of the file. The *eol* option is used to specify the end-of-line comment character as a semicolon (;). Finally, the *tokens* option specifies that tokens 3 through 5 should be placed in iterator variables, starting with *%%C*.

With tokens, you can specify which fields you want to work with in many different ways. Here are some examples:

- **tokens=2,3,7** Use fields 2, 3, and 7.
- **tokens=3-5** Use fields 3, 4, and 5.
- **tokens=*** Examine each line in its entirety and do not break into

fields.

When you work with text files, you should note that all blank lines in text files are skipped and that multiple source files can be specified with wild cards or by entering the file names in a space-separated list, such as

```
for /f "skip=3 eol=; tokens=3-5" %%C in (data1.txt data2.txt) do (
@echo Depart: %%C E-mail: %%D Ext: %%E)
```

If a filename contains a space or you want to execute a command, specify the *usebackq* option and quotation marks, such as

```
for /f "tokens=3-5 usebackq" %%C in ("user data.txt") do (
@echo Depart: %%C E-mail: %%D Ext: %%E)
```

or

```
for /f "tokens=3-5 usebackq" %%C in (`type "user data.txt"`) do (
@echo Depart: %%C E-mail: %%D Ext: %%E)
```

TIP Remember the backquote (`) is used with commands and the single quotation mark (') is used with string literals. In print, these characters no doubt look very similar. However, on a standard keyboard, the backquote (') is on the same key as the tilde (~) and the single quotation mark (') is on the same key as a double quotation mark (").

NOTE In the second example, I use the TYPE command to write the contents of the file to standard output. This is meant to be an example of using a command with the backquote.

Speaking of quotation marks, you use quotation marks when you want to process strings and variable values. Here, you enclose the string or variable name you want to work with in double quotation marks to ensure that the string or variable can be evaluated properly. You do not, however, need to use the *usebackq* option.

Consider the following example:

```
set value=All,Some,None
```

```
for /f "delims=, tokens=1,3" %%A in ("%VALUE%") do (echo %%A %%B)
```

The output is

```
All None
```

Creating Subroutines and Procedures

Normally, the Windows command shell executes scripts line by line, starting at the beginning of the file and continuing until the end of the file. You can change the order of execution. To do this, you use either of the following techniques.

- **Subroutines** With subroutines, you jump to a label within the current script, and execution proceeds to the end of the file.
- **Procedures** With procedures, you call another script and execution of the called script proceeds to the end of its file, and then control returns to the line following the call statement in the original script.

As you can see, the difference between a subroutine and a procedure is primarily in what you want to do. Additionally, while arguments passed in to the script are available in a *goto* subroutine directly, the list of arguments within a called procedure is changed to include the procedure name rather than the script name as argument 0 (*%0*).

Using Subroutines

Subroutines have two parts:

- A *goto* call that specifies the subroutine to which you want to jump
- A label that designates the start of the subroutine

Consider the following subroutine call:

```
if "%1"=="1" goto SUB1
```

Here if the first parameter passed into the script is a 1, the subroutine called *SUB1* is called and the command shell would jump to the corresponding subroutine label. To create a label, you enter a keyword on a line by itself, beginning with a colon, such as

```
:SUB1
```

Although labels can contain just about any valid type of character, you'll usually want to use alphanumeric characters, which make the labels easy to read when you or someone else is going through the code.

When you use *goto*, execution of the script resumes at the line following the target label and continues to the end of the file, unless it's necessary to process any procedure calls or *goto* statements encountered along the way. If the label is before the current position in the script, the command shell can go back to an earlier part of the script. This can create an endless loop (unless there is a control to bypass the *goto* statement). Here's an example of an endless loop:

```
:START
.
.
.
goto START
```

If the label is after the *goto* statement, you can skip commands and jump ahead to a new section of the script, such as

```
goto MIDDLE
.
.
.
:MIDDLE
```

Here, execution of the script jumps to the *:MIDDLE* label and continues to the end of the file. You cannot go back to the unexecuted commands unless you use another *goto* statement.

Sometimes you may not want to execute the rest of the script and instead will want to exit the script after executing subroutine statements. To do this, create an exit label and then go to the exit at the end of the routine, such as

```
goto MIDDLE
.
.
.
:MIDDLE
.
.
.
goto EXIT
.
.
.
:EXIT
```

Listing 3-4 shows a detailed example of working with *goto* and labels. In this example, the value of the script's first parameter determines what subroutine is executed. The first *if* statement handles the case when no parameter is passed in by displaying an error message and exiting. The *goto EXIT* statement following the *if* statements handles the case when an invalid parameter is passed in. Here, the script simply goes to the *:EXIT* label.

LISTING 3-4 Using goto

```
@echo off
if "%1"=="" (echo Error: No parameter passed with script!) & (goto EXIT)
if "%1"=="1" goto SUBROUTINE1
if "%1"=="2" goto SUBROUTINE2
if "%1"=="3" goto SUBROUTINE3
goto EXIT

:SUBROUTINE1
echo In subroutine 1
goto EXIT

:SUBROUTINE2
echo In subroutine 2
goto EXIT

:SUBROUTINE3
echo In subroutine 3
goto EXIT
```

```
:EXIT
echo Exiting...
```

TIP Remember that if the label you call doesn’t exist, you’ll get an error when the end of the file is reached during the search for the nonexistent label, and then the script will exit without executing the other subsequent commands. Old-school command-shell programmers who have been at this for a long time, like me, like to use *goto EXIT* and then provide an actual *:EXIT* label, as shown in the previous example. However, the command interpreter also supports a target label of *:EOF*, which transfers control to the end of the file. This makes *:EOF* an easy way to exit a batch script without defining a label.

Using Procedures

You use procedures to call other scripts without exiting the current script. When you do this, the command shell executes the named script, executing its commands, and then control returns to the original script, starting with the first line following the original call. Consider the following example:

```
if "%1"=="1" call system-checks
if "%1"=="2" call C:\scripts\log-checks
```

CAUTION If you forget to use the *call* statement and reference a script name within a script, the second script executes, but control isn’t returned to the caller.

Here the first call is made to a script expected to be in the current working directory or in the command path. The second call is made to a script with the file path c:\scripts\log-checks.

Any arguments passed to the original script are passed to the called script with one change: The list of arguments is updated to include the procedure name as argument 0 (%0). These procedure-specific arguments remain in effect until the end of the file is reached and control returns to the original script.

You can also pass arguments to the called script, such as

```
set Arg1=mailer1
set Arg2=dc2
set Arg3=web3
call system-checks Arg1 Arg2 Arg3
```

Now within the called script, the variables *Arg1*, *Arg2*, and *Arg3* are available.

Chapter 4. Working with the Registry

The Windows registry stores configuration settings. Using the Reg command-line utility, you can view, add, delete, compare, and copy registry entries. Because the Windows registry is essential to the proper operation of the operating system, make changes to the registry only when you know how these changes will affect the system. Before you edit the registry in any way, perform a complete system backup and create a system recovery data snapshot. This way, if you make a mistake, you can recover the registry and the system.

> *CAUTION* Improperly modifying the Windows registry can cause serious problems. If the registry becomes corrupted, you might have to reinstall the operating system. Double-check the commands you use before executing them. Make sure that they do exactly what you intend.

Understanding Registry Keys and Values

The Windows registry stores configuration settings for the operating system, applications, users, and hardware. Registry settings are stored as keys and values, which are placed under a specific root key controlling when and how the keys and values are used.

Table 5-1 lists the registry root keys as well as a description and the reference name you will use to refer to the root key when working with the REG command. Under the root keys, you'll find the main keys that control system, user, application, and hardware settings. These keys are organized into a tree structure, with folders representing keys. For example, under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services, you'll find folders for all services installed on the system. Within these folders are the registry keys that store important service configuration settings and their subkeys.

TABLE 5-1 Keys in the Windows Registry

ROOT KEY	REFERENCE NAME	DESCRIPTION
HKEY_CURRENT_USER	HKCU	Stores configuration settings for the current user.
HKEY_LOCAL_MACHINE	HKLM	Stores system-level configuration settings.
HKEY_CLASSES_ROOT	HKCR	Stores configuration settings for applications and files. Also ensures that the correct application is opened when a file is accessed.
HKEY_USERS	HKU	Stores default-user and other-user settings by profile.
HKEY_CURRENT_CONFIG	HKCC	Stores information about the hardware profile being used.

Keys that you want to work with must be designated by their folder path. For example, the path to the DNS key is HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DNS and, using the abbreviated path HKLM\SYSTEM\CurrentControlSet\Services\DNS, you can view and manipulate this key.

Key values are stored as a specific data type. Table 5-2 provides a summary of the main data types used with keys.

TABLE 5-2 Registry Key Values and Data Types

DATA TYPE	DESCRIPTION	EXAMPLE
REG_BINARY	Identifies a binary value. Binary values are stored using base-2 (0 or 1 only) but are displayed and entered in hexadecimal (base-16) format.	01 00 14 80 90 00 00 9c 00
REG_DWORD	Identifies a binary data type in which 32-bit integer values are stored as four byte-length values in hexadecimal.	0x00000002
REG_EXPAND_SZ	Identifies an expandable string value, which is usually used with directory paths.	%SystemRoot%\dns.exe
REG_MULTI_SZ	Identifies a multiple string value.	Tcpip Afd RpcSc
REG_NONE	Identifies data without a particular type. This data is written as binary values but displayed and entered in hexadecimal (base-16) format.	23 45 67 80
REG_SZ	Identifies a string value containing a sequence of characters.	DNS Server

As long as you know the key path and understand the available key data types, you can use the REG command to view and manipulate keys in a types, you can use the REG command to view and manipulate keys in a variety of ways. REG has several different subcommands, and we'll explore some of them. The sections that follow discuss each of the following REG subcommands:

- **REG add** Adds a new subkey or entry to the registry
- **REG delete** Deletes a subkey or entries from the registry
- **REG query** Lists the entries under a key and the names of subkeys (if any)
- **REG compare** Compares registry subkeys or entries
- **REG copy** Copies a registry entry to a specified key path on a local or remote system
- **REG flags** Displays and manages the current flags of a specified key
- **REG restore** Writes saved subkeys, entries, and values back to the registry
- **REG save** Saves a copy of specified subkeys, entries, and values to a file

The following sections will also discuss the following commands for performing advanced registry manipulation:

- **REG import** Imports a specified hive file into the registry
- **REG export** Exports specified subkeys, entries, and values to a registry file
- **REG load** Loads a specified hive file into the registry
- **REG unload** Unloads a specified hive file into the registry

> *NOTE* The REG command is run using the permissions of the current user. If you want to use a different set of permissions, the easiest way is to log on as that user.

Querying Registry Values

Using REG query, you can read registry values by referencing the full path

Using REG query, you can read registry values by referencing the full path and name of a key or key value that you want to examine. The basic syntax is

```
reg query KeyName [/v ValueName]
```

where *KeyName* is the name of the key you want to examine and *ValueName* is an optional parameter that specifies a specific key value. In the following example, you query the DNS key under the current control set:

```
reg query HKLM\SYSTEM\CurrentControlSet\Services\DNS
```

Alternatively, if you know the specific key value you want to examine, you can limit the query results using the /V parameter. In this example, you list the value of the ImagePath entry for the DNS key:

```
reg query HKLM\SYSTEM\CurrentControlSet\Services\DNS /v ImagePath
```

The key path can also include the UNC name or IP address of a remote computer that you want to examine, such as \\Mailer1 or \\192.168.1.100. However, keep in mind that on a remote computer, you can only work with the HKLM and HKU root keys. In this example, you examine the DNS key on MAILER1:

```
reg query \\Mailer1\HKLM\SYSTEM\CurrentControlSet\Services\DNS
```

> *NOTE* If you specify a nonexistent key or value, an error message is displayed. Typically, it reads: ERROR: The system was unable to find the specified registry key or value.

Comparing Registry Keys

With REG compare, you can compare registry entries and values between two systems or between two different keys on the same system. Performing registry comparisons is useful in the following situations:

- **When you are trying to troubleshoot service and application**

configuration issues At such times, it is useful to compare the registry configurations between two different systems. Ideally, these systems include one that appears to be configured properly and one that you suspect is misconfigured. You can then perform a comparison of the configuration areas that you suspect are causing problems.

- **When you want to ensure that an application or service is configured the same way on multiple systems** Here you would use one system as the basis for testing the other system configurations. Ideally, the basis system is configured exactly as expected before you start comparing its configuration to other systems.

The basic syntax for REG compare is

```
reg compare KeyName1 KeyName2 [/v ValueName]
```

where *KeyName1* and *KeyName2* are the names of the subkeys that you want to compare and *ValueName* is an optional parameter that specifies a specific key value to compare. The key name can include the UNC name or IP address of a remote computer that you want to examine. In the following example, you compare the DNS key under the current control set on MAILER1 and MAILER2:

```
reg compare \\Mailer1\HKLM\SYSTEM\CurrentControlSet\Services\DNS
\\Mailer2\HKLM\SYSTEM\CurrentControlSet\Services\DNS
```

If the keys are configured the same, the output is

```
Results Compared: Identical
The operation completed successfully.
```

If the keys are configured differently, the output shows the differences. Any differences that begin with the < character pertain to the first key specified and differences that begin with the > character pertain to the second key specified. The output will also state

```
Results Compared: Different
The operation completed successfully.
```

> *TIP* Differences are displayed because the *Od parameter is assumed by default. Using additional parameters, you can also specify that you want to see all differences and matches (*Oa), only matches (/Os), or no results (/On).

Additionally, if you want to compare all subkeys and entries recursively, you can add the /S parameter, as shown in the following example:

```
reg compare \\Mailer1\HKLM\SYSTEM\CurrentControlSet\Services\DNS
\\Mailer2\HKLM\SYSTEM\CurrentControlSet\Services\DNS /s
```

Now the key, all subkeys, and all related entries for the DNS key on MAILER1 and MAILER2 are compared.

Saving and Restoring Registry Keys

Before you modify registry entries, it is a good idea to save the keys you will use. If anything goes wrong, you can restore those keys to their original settings. To save a copy of a registry subkey and all its related subkeys and values, use REG save, as shown here:

```
reg save KeyName "FileName"
```

where KeyName is the path to the subkey you want to save and *FileName* is the text name of the registry hive file you want to create. The subkey path can include the UNC name or IP address of a remote computer. However, on a remote computer, you can only work with the HKLM and HKU root keys. Additionally, the filename should be enclosed in double quotation marks and should end in the .hiv extension to indicate it is a registry hive file, as shown in the following example:

```
reg save HKLM\SYSTEM\CurrentControlSet\Services\DNS "DNSKey.hiv"
```

Here, you are saving the DNS subkey and its related subkeys and values to the file named Dnskey.hiv. The filename can also include a directory path, as shown in this example:

```
reg save \HKLM\SYSTEM\CurrentControlSet\Services\DNS
"\\Mailer1\SavedData\DNSKey.hiv"
```

If the registry hive file exists, you will be prompted to overwrite the file. Press Y to overwrite. If you want to force overwrite without prompting, use the /Y parameter.

To restore a registry key that you saved previously, use Reg restore. The syntax for REG restore is

```
reg restore KeyName "FileName"
```

where *KeyName* is the path to the subkey you want to save and *FileName* is the text name of the registry hive file you want to use as the restore source. Unlike REG copy, REG restore can be used only on a local computer, meaning you cannot restore registry keys on a remote computer using the command. You can, however, start a remote desktop session on the remote computer and then use the remote desktop logon to restore the registry key on the local computer.

An example using REG restore is shown here:

```
reg restore HKLM\SYSTEM\CurrentControlSet\Services\DNS "DNSKey.hiv"
```

Here, you are restoring the DNS key saved previously to the DNSKey.hiv file.

Adding Registry Keys

To add subkeys and values to the Windows registry, use REG add. The basic syntax for creating a key or value is

```
reg add KeyName /v ValueName t DataType d Data
```

where *KeyName* is the name of the key you want to examine, *ValueName* is the subkey or key value to create, *DataType* is the type of data, and *Data* is the actual value you are inserting. That seems like a lot of values, but it is fairly straightforward. Consider the following example:

```
reg add HKLM\SYSTEM\CurrentControlSet\Services\DNS /v DisplayName
/t REG_SZ /d "DNS Server"
```

Here, you add a key value called DisplayName to the DNS key in the registry. The key entry is a string with the “DNS Server” value. Note the double-quotation marks. The quotation marks are necessary in this example because the string contains a space. If the key or value you are attempting to add already exists, you are prompted to overwrite the existing data. Enter Y to overwrite, or N to cancel. To force overwriting an existing registry key or value without a prompt, use the /F parameter.

When you set expandable string values (REG_EXPAND_SZ), you must use the caret (^) to escape the percent symbols (%) that designate the environment variable you use. Consider the following example:

```
reg add HKLM\SYSTEM\CurrentControlSet\Services\DNS /v ImagePath
/t REG_EXPAND_SZ /d ^%SystemRoot^%\System32\dns.exe
```

Here, you enter **^%SystemRoot^%** so that the SystemRoot environment variable is properly entered and interpreted.

When you set non-string values, you don’t need to use quotation marks, as shown in this example:

```
reg add HKLM\SYSTEM\CurrentControlSet\Services\DNS /v ErrorControl
t REG_DWORD d 0x00000001
```

Copying Registry Keys

Using REG copy, you can copy a registry entry to a new location on a local or remote system. The basic syntax for REG copy is

```
reg copy KeyName1 KeyName2
```

where *KeyName1* is the path to the subkey you want to copy and *KeyName2* is the path to the subkey destination. Although the subkey paths can include the UNC name or IP address of a remote computer, REG copy is limited in scope with regard to which root keys you can use when working with remote source or destination keys, as follows:

- A remote source subkey can use only the HKLM or HKU root keys.
- A remote destination subkey can use only the HKLM or HKU root keys.

In the following example, you copy the DNS subkey on the local system to the DNS subkey on MAILER2:

```
reg copy HKLM\SYSTEM\CurrentControlSet\Services\DNS
    \\Mailer2\HKLM\SYSTEM\CurrentControlSet\Services\DNS
```

By adding the /S parameter, you can copy the specified subkey as well as all subkeys and key entries under the specified subkey. In this example, the DNS subkey and all related subkey and values are copied:

```
reg copy HKLM\SYSTEM\CurrentControlSet\Services\DNS
    \\Mailer2\HKLM\SYSTEM\CurrentControlSet\Services\DNS /s
```

If values exist at the destination path, REG copy will prompt you to confirm that you want to overwrite each existing value. Press Y or N as appropriate. You can also press A to overwrite all existing values without further prompting.

> *NOTE* If you don’t want prompts to be displayed, you can use the /F parameter to force overwrite without prompting. However, before you copy over an existing registry key, you may want to save the key so that it can be restored if problems occur. To do this, use REG save and REG restore as discussed earlier in the section of this chapter titled “Saving and Restoring Registry Keys.”

Deleting Registry Keys

To delete subkeys and values from the Windows registry, use REG delete.

REG delete has several different syntaxes. If you want to delete a subkey and all subkeys and entries under the subkey, use the following syntax:

```
reg delete KeyName
```

where *KeyName* is the name of the subkey you want to delete. Although the subkey path can include the UNC name or IP address of a remote computer, a remote source subkey can use only the HKLM or HKU root keys. Consider the following example:

```
reg delete \\Mailer1\HKLM\SYSTEM\CurrentControlSet\Services\DNS2
```

Here you delete the DNS2 subkey and all subkeys and entries under the subkey on MAILER1.

If you want to limit the scope of the deletion, specify that only a specific entry under the subkey should be deleted using the following syntax:

```
reg delete KeyName /v ValueName
```

where *KeyName* is the name of the subkey you want to work with and *ValueName* is the name of the specific entry to delete. As before, the subkey path can include the UNC name or IP address of a remote computer. However, a remote source subkey can use only the HKLM or HKU root keys. In this example, you delete the Description entry for the DNS2 subkey on MAILER2:

```
reg delete \\Mailer2\HKLM\SYSTEM\CurrentControlSet\Services\DNS2 /v
Description
```

> *TIP* In both cases, you will be prompted to confirm that you want to delete the specified entry permanently. Press Y to confirm the deletion. You can force deletion without prompting using the /F parameter. Another useful parameter is *Va. Using the* Va parameter, you can specify that only values under the subkey should be deleted. In this way, subkeys under the designated subkey are not deleted.

Exporting and Importing Registry Keys

Sometimes you might find it necessary or useful to copy all or part of the registry to a file and then use this copy on another computer. For example, if you've installed a component that requires extensive configuration, you might want to use it on another computer without having to go through the whole configuration process again. To do this, you would install and configure the component, export the component's registry settings from the computer, copy the settings to another computer, and then import the registry settings so that the component is properly configured. Of course, this technique works only if the complete configuration of the component is stored in the registry, but you can see how useful being able to export and import registry data can be.

When you use the REG export and REG import commands, exporting and importing registry data is fairly easy. This includes branches of data stemming from a particular root key as well as individual subkeys and the values they contain. When you export data, you create a .reg file that contains the designated registry data. This registry file is a script that can then be loaded back into the registry of this or any other computer by importing it.

Because the registry script is written as standard text, you could view it and, if necessary, modify it in any standard text editor as well. To export registry data to a file in the current directory, use the following syntax:

```
reg export KeyName FileName
```

where *KeyName* is the name of the subkey you want to work with and *FileName* is the name of the file in which to store the registry data. As before, the subkey path can include the UNC name or IP address of a remote computer. However, a remote source subkey can use only the HKLM or HKU root keys. In this example, you export the MSDTC subkey on MAILER1:

```
reg export \\Mailer1\HKLM\SOFTWARE\Microsoft\MSDTC msdtc-regkey.reg
```

You can export keys at any level of the registry. For example, you export the HKLM root key and all its subkeys using the following command line:

```
reg export HKLM hklm.reg
```

TIP Add the /Y parameter to force REG export to overwrite an existing file. You can export the entire registry at the command line by typing **regedit /e SaveFile**, where SaveFile is the complete file path to the location where you want to save the copy of the registry. For example, if you wanted to save a copy of the registry to C:\Save\Regdata.reg, you would type **regedit /e C:\Save\Regdata.reg**.

Importing registry data adds the contents of the registry script file to the registry of the computer you are working with, either creating new keys and values if they didn't previously exist or overwriting keys and values if they did previously exist. You can import registry data using the REG import command and the following syntax:

```
reg import FileName
```

where *FileName* is the name of the registry file in the current directory you want to import, such as:

```
reg import msdtc-regkey.reg
```

You cannot perform imports remotely or use non-local files for imports. When you are importing registry keys, you must be logged on locally to the computer and the file must exist on the local computer.

Loading and Unloading Registry Keys

Just as you sometimes must export or import registry data, you'll sometimes need to work with individual hive files. The most common reason for doing this is when you must modify a user's profile to correct an issue that prevents the user from accessing or using a system. For example, you may need to load and modify the settings for a user profile because the user inadvertently changed the display mode to an invalid setting and can no longer access the computer locally. With the user profile data loaded into the registry, you could edit the registry to correct the problem and then save the changes so that the user can once again log on to the system.

Another reason for loading registry keys is to change a particular part of the registry on a remote system. Loading and unloading hives affects only HKEY_LOCAL_MACHINE and HKEY_USERS, and you can perform these actions only when one of these root keys is selected. Rather than replacing the selected root key, the hive you are loading then becomes a subkey of that root key. HKEY_LOCAL_MACHINE and HKEY_USERS are of course used to build all the logical root keys used on a system, so you could in fact work with any area of the registry.

The file to be loaded must have been saved by the REG save command. You can load a previously saved hive file using the REG load command and the following syntax:

```
reg load RootKey\KeyName FileName
```

where *RootKey* is the root key under which the temporary key will be created, *KeyName* is the name of the temporary subkey you want to create, and *FileName* is the name of the saved hive file to load. You must create the temporary subkey under HKLM or HKU. In the following example, you create a temporary key called CurrTemp under HKLM and load the Working.hiv hive file into this key:

```
reg load HKLM\CurrTemp Working.hiv
```

You cannot perform loads remotely or use non-local files for loads. When you are loading registry keys, you must be logged on locally to the computer and the file must exist on the local computer.

Once you load a registry key, you can manipulate its subkeys and values using the techniques discussed previously. When you are finished modifying the key, you can save the key to a new registry file using REG save. After you save the key, you can unload the hive file and remove it from the computer’s memory and the working registry by using the REG unload command and the following syntax:

```
reg unload RootKey\KeyName
```

where *RootKey* is the root key under which the temporary key was created and *KeyName* is the name of the temporary subkey you want to unload. In the following example, you unload the temporary key called CurrTemp under HKLM:

```
reg unload HKLM\CurrTemp
```

> *NOTE* You can't work with hive files that are already being used by the operating system or another process. You can, however, make a copy of the hive and then work with it. At the command line, type **reg save** followed by the abbreviated name of the root key to save and the filename to use for the hive file. For example, type **reg save hkcu c:\currhkcu.hiv** to save HKEY_LOCAL_MACHINE to a file called Currhkcu.hiv on the root folder of drive C. Although you can save the logical root keys (HKCC, HKCR, HKCU) in this manner, you can save only subkeys of HKLM and HKU for use in this technique.

Following these rules, if you needed to repair an area of the registry on a remote computer, you could:

1. Access the remote computer and save the registry hive to a file using the REG save command.
2. Copy the registry file to a folder on your computer using XCOPY or a similar command.
3. Load the related hive file into the registry of your computer using the REG load command.
4. Make any necessary changes and then save the changes using the REG save command.
5. Import the registry hive on the remote computer to repair the problem using the REG import command.
6. After you test the changes on the remote computer, unload the registry hive from your computer using the REG unload command.

Chapter 5. Managing System Services

Services provide key functions to workstations and servers. To manage system services on local and remote systems, you'll use the service controller command SC, which has several subcommands, only some of which are explored here. The sections that follow discuss each of these subcommands:

- **SC config** Configures service startup and logon accounts
- **SC query** Displays the list of all services configured on the computer
- **SC qc** Displays the configuration of a specific service
- **SC start** Starts services
- **SC stop** Stops services
- **SC pause** Pauses services
- **SC continue** Resumes services
- **SC failure** Sets the actions to take upon failure of a service
- **SC qfailure** Views the actions to take upon failure of a service

With all commands, you can specify the name of the remote computer whose services you want to work with. To do this, insert the UNC name or IP address of the computer before the subcommand you want to use. This makes the syntax `sc ServerName Subcommand`

Viewing Configured Services

To get a list of all services configured on a system, type the following command at the command prompt: `sc query type= service state= all` or

`sc` *ServerName* `query type= service state= all` where *ServerName* is the UNC name or IP address of the remote computer, such as \\Mailer1 or \\192.168.1.100, as shown in the following examples:

```
sc \\Mailer1 query type= service state= all
sc \\192.168.1.100 query type= service state= all
```

NOTE You must

include a space after the equal sign (=) as used with type= *service* and *state*= *all*. If you don't use a space, the command will fail.

With the *state* flag, you can also use the value active (to show running services only) or *inactive* (to show all paused or stopped services). Consider the following examples: `sc \\Mailer1 query type= service state= active`

`sc \\Mailer1 query type= service state= inactive` In the first example, you query MAILER1 for a list of all services that are running. In the second example, you query MAILER1 for a list of all services that are stopped.

The output of SC query shows the services and their configurations. Each service entry is formatted as follows:

```
SERVICE_NAME: W3SVC
DISPLAY_NAME: World Wide Web Publishing Service
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

The fields you will work with the most are

- **Service Name** The abbreviated name of the service. Only services installed on the system are listed here. If a service you need isn't listed, you'll need to install it.
- **Display Name** The descriptive name of the service.
- **State** The state of the service as Running, Paused, or Stopped.

As you'll see if you run the SC query command, the output is very long and is best used with a filter to get only the information you want to see. For example, if you use the following command, you clean up the output to show only the most important fields: `sc query type= service | find /v "x0"`

Here you pipe the output of SC query through the FIND command and clean up the output so the service entries appear, as shown in this example:

```
SERVICE_NAME: W3SVC
```

```
DISPLAY_NAME: World Wide Web Publishing Service
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                       (STOPPABLE, PAUSABLE, ACCEPTS_SHUTDOWN)
```
NOTE The parameter / V "x0" tells the FIND command to display only lines of output that do not contain the text x0, which is the common text on the WIN32_Exit_Code, Service_Exit_Code, Checkpoint, and Wait_Hint fields. By specifying that you don't want to see lines of output that contain this value, you therefore remove these unwanted fields from the display.

If you know the name of a service you want to work with, you can use SC qc to display its configuration information. The syntax is `sc qc ServiceName` where *ServiceName* is the name of the service you want to examine. The output for individual services looks like this:

```
SERVICE_NAME: w3svc
        TYPE               : 20  WIN32_SHARE_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\WINDOWS\System32\svchost.exe -k iissvcs
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : World Wide Web Publishing Service
        DEPENDENCIES       : RPCSS
                           : HTTPFilter
                           : IISADMIN
        SERVICE_START_NAME : LocalSystem
```
Note that the output doesn't tell you the current status of the service. It does, however, tell you the following:

- **Binary Path Name** The file path to the executable for the service
- **Dependencies** Services that cannot run unless the specified service is running
- **Display Name** The descriptive name of the service
- **Service Start Name** The name of the user account the service logs on as
- **Start Type** The startup configuration of the service

NOTE Services that are configured to start automatically are listed as AUTO_START. Services that are configured to start manually are listed as DEMAND_START. Services that are disabled are listed as DISABLED.

- **Type** The type of service and whether it is a shared process

NOTE When you are configuring a service logon, it is sometimes important to know whether a process runs in its own context or is shared. Shared processes are listed as WIN32_SHARE_PROCESS. Processes that run in their own context are listed as WIN32_OWN_PROCESS.

Starting, Stopping, and Pausing Services

When troubleshooting system problems, you'll often have to start, stop, or pause Windows services. The related SC commands and their syntaxes are Start a service:

`sc start ServiceName` Pause a service:

`sc pause ServiceName` Resume a paused service: `sc continue` *ServiceName* Stop a service:

`sc stop ServiceName` where *ServiceName* in each case is the abbreviated name of the service you want to work with, such as `sc start w3svc` As with all SC commands, you can also specify the name of the remote computer whose services you want to work with. For example, to start the w3svc on MAILER1, you would use the following command: `sc \\Mailer1 start w3svc` The state listed in the results should show START_PENDING. With stop, pause, and continue you'll see STOP_PENDING, PAUSE_PENDING, and CONTINUE_PENDING respectively as well. If an error results, the output states FAILED and error text is provided to describe the reason for the failure in more detail. If you are trying to start a service that is already started, you'll see the error `An instance of the service is already running.`

If you are trying to stop a service that is already stopped, you'll see the error `The service has not been started.`

Configuring Service Startup

You can set Windows services to start manually or automatically. You can also turn them off permanently by disabling them. You configure service startup using `sc config` *ServiceName* `start=` *flag* where *ServiceName* is the abbreviated name of the service you want to work with and *flag* is the startup type to use. For services, valid flag values are

- **Auto** Starts service at system startup
- **Demand** Allows the services to be started manually
- **Disabled** Turns off the service
- **Delayed-Auto** Delays the start of the service until all non-delayed automatic services have started

Following this, you can configure a service to start automatically by using `sc config w3svc start= auto` or

`sc \\Mailer1 config w3svc start= auto` *NOTE* You must include a space after the equal sign (=) as used with *start= auto*. If you don't use a space, the command will fail. Note also the command only reports SUCCESS or FAILURE. It won't tell you that the service was already configured in the startup mode you've specified.

Disabling a service doesn't stop a running service. It only prevents it from being started the next time the computer is booted. To ensure that the service is disabled and stopped, run SC stop and then SC config.

Configuring Service Logon

You can configure Windows services to log on as a system account or as a specific user. To ensure a service logs on as the LocalSystem account, use `sc config` *ServiceName* `obj= LocalSystem` where *ServiceName* is the name of the service you are configuring to use the LocalSystem account. If the service provides a user interface that can be manipulated, add the flags **type= interact type= own**, as shown in the following example: `sc config w3svc obj= LocalSystem type= interact type= own` The type= interact flag specifies that the service is allowed to interact with the Windows desktop. The type= own flag specifies that the service runs in its own process. In the case of a service that shares its executable files with other services, you would use the type= share flag, as shown in this example: `sc config w3svc obj= LocalSystem type= interact type= share` *TIP* If you don't know whether a service runs as a shared process or in its own context, use SC qc to determine the service's start type. This command is discussed in the section titled "Viewing Configured Services," earlier in this chapter.

Services can also log on using named accounts. To do this, use `sc config` *ServiceName* `obj= [`*Domain*`\]User password=` *Password* where *Domain* is the optional domain name in which the user account is located, User is the name of the user account whose permissions you want to use, and *Password* is the password of that account. Consider the following example: `sc config w3svc obj= imagnedl\webbies password= blue5!CraZy` Here, you configure W3svc to use the Webbies account in the IMAGINEDL domain. The output of the command should state SUCCESS or FAILED. The change will fail if the account name is invalid or doesn't exist, or if the password for the account is invalid.

> *NOTE* If a service has been previously configured to interact with the desktop under the LocalSystem account, you cannot change the service to run under a domain account without using the type= own flag. The syntax therefore becomes sc config **ServiceName** obj= [**Domain**\]**User** password= Password type= own.

> *REAL WORLD* As an administrator, you should keep track of any accounts that are used with services. These accounts can be the source of huge security problems if they're not configured properly. Service accounts should have the strictest security settings and as few permissions as possible while allowing the service to perform

necessary functions. Typically, accounts used with services don't need many of the permissions you would assign to a normal user account. For example, most service accounts don't need the right to log on locally. Every administrator should know what service accounts are used (so that he or she can better track use of these accounts), and the accounts should be treated as if they were administrator accounts. This means secure passwords, careful monitoring of account usage, careful application of account permissions and privileges, and so on.

Configuring Service Recovery

Using the SC failure command, you can configure Windows services to take specific actions when a service fails. For example, you can attempt to restart the service or run an application.

You can configure recovery options for the first, second, and subsequent recovery attempts. The current failure count is incremented each time a failure occurs. You can also set a parameter that specifies the time that must elapse before the failure counter is reset. For example, you could specify that if 24 hours have passed since the last failure, the failure counter should be reset.

Before you try to configure service recovery, check the current recovery settings using SC qfailure. The syntax is `sc qfailure` *ServiceName* where *ServiceName* is the name of the service you want to work with, such as `sc qfailure w3svc` You can of course specify a remote computer as well, such as `sc \\Mailer1 qfailure w3svc` or

`sc \\192.168.1.100 qfailure w3svc` In the output, the failure actions are listed in the order they are performed. In the following example output, W3svc is configured to attempt to restart the service the first and second time the service fails and to restart the computer if the service fails a third time: `[SC] QueryServiceConfig2 SUCCESS`

```
SERVICE_NAME: w3svc
```

```
RESET_PERIOD (in seconds) : 86400
REBOOT_MESSAGE            :
COMMAND_LINE              :
FAILURE_ACTIONS          : RESTART -- Delay = 1 milliseconds.
                           RESTART -- Delay = 1 milliseconds.
                           REBOOT -- Delay = 1000 milliseconds.
```

NOTE Windows automatically configures recovery for some critical system services during installation. Typically, these services are configured so that they attempt to restart the service. A few services are configured so that they run programs. For example, the IIS Admin service is configured to run a program called Iisreset.exe if the service fails. This program is an application that corrects service problems and safely manages dependent IIS services while working to restart the IIS Admin service.

The command you use to configure service recovery is SC failure and its basic syntax is `sc failure ServiceName reset= FailureResetPeriod actions= RecoveryActions` where *ServiceName* is the name of the service you are configuring, *FailureResetPeriod* specifies the time, in seconds, that must elapse without failure in order to reset the failure counter, and *RecoveryActions* are the actions to take when failure occurs plus the delay time (in milliseconds) before that action is initiated. The available recovery actions are

- **Take No Action (indicated by an empty string "")** The operating system won’t attempt recovery for this failure but might still attempt recovery of previous or subsequent failures.
- **Restart The Service** Stops and then starts the service after a brief pause.
- **Run A Program** Allows you to run a program or a script in case of failure. The script can be a batch program or a Windows script. If you select this option, set the full file path to the program you want to run and then set any necessary command-line parameters to pass in to the program when it starts.
- **Reboot The Computer** Shuts down and then restarts the computer after the specified delay time is elapsed.

BEST PRACTICES When you configure recovery options for critical services, you might want to try to restart the service on the first and second attempts and then reboot the server on the third attempt.

When you work with SC failure, keep the following in mind:

- **The reset period is set in seconds**. Reset periods are commonly set in multiples of hours or days. An hour is 3,600 seconds and a day is 86,400 seconds. For a two-hour reset period, for example, you'd use the value 7,200.
- **Each recovery action must be followed by the time to wait (in milliseconds) before performing the action**. For a service restart you'll probably want to use a short delay, such as 1 millisecond (no delay), 1 second (1,000 milliseconds), or 5 seconds (5,000 milliseconds). For a restart of the computer, you'll probably want to use a longer delay, such as 15 seconds (15,000 milliseconds) or 30 seconds (30,000 milliseconds).
- **Enter the actions and their delay times as a single text entry with each value separated by a forward slash (/).** For example, you could use the value: restart/1000/restart/1000/reboot/15000. Here, on the first and second attempts the service is restarted after a 1-second delay, and on the third attempt the computer is rebooted after a 15-second delay.

Consider the following examples: `sc failure w3svc reset= 86400 actions= restart/1/restart/1/reboot/30000`

Here, on the first and second attempts the service is restarted almost immediately, and on the third attempt the computer is rebooted after a 30-second delay. In addition, the failure counter is reset if no failures occur in a 24-hour period (86,400 seconds). You can also specify a remote computer by inserting the UNC name or IP address as shown in previous examples.

If you use the Run action, you specify the command or program to run using the *Command=* parameter. Follow the *Command=* parameter with the full file path to the command to run and any arguments to pass to the command. Be sure to enclose the command path and text in double quotation marks, as in the following example: `sc failure w3svc reset= 86400 actions= restart/1/restart/1/run/30000 command= "c:\restart_w3svc.exe 15"`

Chapter 6. Restarting and Shutting Down Computers from the Command Line

You'll often find that you need to shut down or restart systems. One way to do this is to use the Shutdown utility, which you can use to work with both local and remote systems. Another way to manage system shutdown or restart is to schedule a shutdown. Here, you can use Schtasks to specify when shutdown should be run or you can create a script with a list of shutdown commands for individual systems.

REAL WORLD Although Windows systems usually start up and shut down without problems, they can occasionally stop responding during these processes. If this happens, try to determine the cause. Some of the reasons systems might stop responding include the following:

1. The system is attempting to execute or is running a startup or shutdown script that has not completed or is itself not responding (and in this case, the system might be waiting for the script to time out).
2. A startup initialization file or service may be the cause of the problem and if so, you might need to troubleshoot startup items using the System Configuration Utility (Msconfig). Disabling a service, startup item, or entry in a startup initialization file might also solve the problem.
3. The system may have an antivirus program that is causing the problem. In some cases, the antivirus program may try to scan the removable media drives when you try to shut down the system. To resolve this, configure the antivirus software so that it doesn't scan the removable media drives or other drives with removable media on shutdown. You could also try temporarily disabling or turning off the antivirus program.
4. Improperly configured sound devices can cause startup

and shutdown problems. To determine what the possible source is, examine each of these devices in turn. Turn off sound devices and then restart the computer. If the problem clears up, you have to install new drivers for the sound devices you are using or you may have a corrupted Start Windows or Exit Windows sound file.

5. Improperly configured network cards can cause startup and shutdown problems. Try turning off the network adapter and restarting. If that works, you might need to remove and then reinstall the adapter's driver or obtain a new driver from the manufacturer.
6. Improperly configured video adapter drivers can cause startup and shutdown problems. From another computer, remotely log on and try to roll back the current video drivers to a previous version. If that's not possible, try uninstalling and then reinstalling the video drivers.

Managing Restart and Shutdown of Local Systems

On a local system, you can manage shutdown and restart using the following commands: Shutdown local system:

`shutdown s t` *ShutdownDelay* `l f` Restart local system:

`shutdown r t` *ShutdownDelay* `l f` Cancel delayed shutdown of local computer:

`shutdown /a`

where /T *ShutdownDelay* is used to set the optional number of seconds to wait before shutdown or restart, *L optionally logs off the current user immediately, and* F optionally forces running applications to close without warning users in advance. In this example, the local system is restarted after a 60-second delay: `shutdown r t 60`

BEST PRACTICES In most network environments, system uptime is of the utmost importance. Systems that are restarting or shutting down aren't available to users, which might mean someone won't be able to finish her work and might get upset as a result. Rather than shut down systems in the middle of business hours, consider performing shutdowns before or after normal business hours. But if you need to shut down a system during business hours, warn users beforehand if possible, allowing them to save current work and log off the system as necessary.

Managing Restart and Shutdown of Remote Systems

With remote systems, you need to specify the UNC name or IP address of the system you want to shut down or restart using the /M parameter. Thus, the basic syntax for shutdown, restart, and cancel delayed shutdown needs to be modified as shown in these examples: Shutdown remote system:

`shutdown s t` *ShutdownDelay* `l f /m \\System` Restart remote system:

`shutdown r t` *ShutdownDelay* `l f /m \\System` Cancel delayed shutdown of remote computer:

`shutdown /a /m \\`*System* In this example, MAILER1 is restarted after a 30-second delay: `shutdown r t 30 /m \\Mailer1`

In this example, the system with the IP address 192.168.1.105 is restarted immediately and running applications are forced to stop running: `shutdown r f /m \\192.168.1.105`

Adding Shutdown or Restart Reasons and Comments

In most network environments, it's a good idea to document the reasons for shutting down or restarting computers. Following an unplanned shutdown, you can document the shutdown in the computer's system log by expanding the syntax to include the following parameters: `/e c`

`"UnplannedReason" d MajorCode:MinorCode` where *E replaces the* R switch, */C* "*UnplannedReason*" sets the detailed reason (which can be up to 512 characters in length) for the shutdown or restart, and /D *MajorCode:MinorCode* sets the reason code for the shutdown. Reason codes are arbitrary, with valid major codes ranging from 0 to 255 and valid minor reason codes ranging from 0 to 65,535.

For a planned restart, consider the following example: `shutdown r m \\Mailer1 c "System Reset" d 5:15`

In this example, you are restarting MAILER1 and documenting the reason for the unplanned restart as a "System Reset" using the reason code 5:15.

Table 5-3 summarizes the common reasons and codes for shutdowns and restarts for Windows 8.1, Windows Server 2012 and Windows 2012 R2. As the table shows, Windows can generate the prefix code E for Expected, U for Unexpected, and P for planned as well as various combinations of these prefix codes.

TABLE 5-3 Common Reasons and Codes for Shutdowns and Restarts

PREFIX CODE	**MAJOR CODE**	**MINOR CODE**	**SHUTDOWN OR RESTART TYPE**
U	0	0	Other (Unplanned)
E	0	0	Other (Unplanned)
E P			

0

0

Other (Planned)

U

0

5

Other Failure: System Unresponsive E

1

1

Hardware: Maintenance (Unplanned) E P

1

1

Hardware: Maintenance (Planned)

E

1

2

Hardware: Installation (Unplanned) E P

1

2

Hardware: Installation (Planned) P

2

3

Operating System: Upgrade (Planned) E

2

4

Operating System: Reconfiguration (Unplanned) E P

2

4

Operating System: Reconfiguration (Planned) P

2

16

Operating System: Service pack (Planned) U

2

17

Operating System: Hotfix (Unplanned) P

2

17

Operating System: Hotfix (Planned) U

2

18

Operating System: Security fix (Unplanned) P

2

18

Operating System: Security fix (Planned) E

4

1

Application: Maintenance (Unplanned) E P

4

1

Application: Maintenance (Planned) E P

4

2

Application: Installation (Planned) E

4

5

Application: Unresponsive

E

4

6

Application: Unstable

U

5

15

System Failure: Stop error

E

5

19

Security issue

U

5

19

Security issue

E P

5

19

Security issue (Planned)

E

5

20

Loss of network connectivity (Unplanned) U

6

11

Power Failure: Cord Unplugged

U

6

12

Power Failure: Environment

P

7

0

Legacy API shutdown (Planned)

For the SHUTDOWN command, only the P: and the U: prefixes are accepted. For example, with planned shutdowns and restarts, prefix the reason codes with p: to indicate a planned shutdown, as shown here: `c "PlannedReason" d p:MajorCode:MinorCode` For instance, consider the following code:

```
shutdown r m \\Mailer1 c "Planned Application Upgrade" d p:4:2
```

In this example, you are restarting MAILER1 and documenting the reason for the planned restart as a “Planned Application Upgrade” using the reason code 4:2.

Chapter 7. Event Logging, Tracking, and Monitoring

Up to this point, we have focused on tools and techniques used to manage local and remote systems from the command line. Now let’s look at how the event logs can be used for monitoring and optimization. Monitoring is the process by which systems are regularly checked for problems. Optimization is the process of fine-tuning system performance to maintain or achieve its optimal capacity.

This chapter examines the logging tools available for Windows systems that can help you to identify and track system problems, monitor applications and services, and maintain system security. When systems slow down, behave erratically, or experience other problems, you may want to look to the event logs to identify the potential source of the problem. Once you’ve identified problem sources or issues, you can perform maintenance or preventative tasks to resolve or eliminate them. Using performance monitoring, you can watch for events to occur and take appropriate action to resolve them.

Windows Event Logging

In Microsoft Windows, an event is any significant occurrence in the operating system that requires users or administrators to be notified. Events are recorded in the Windows event logs and provide important historical information to help you monitor systems, maintain system security, solve problems, and perform diagnostics. It’s not just important to sift regularly through the information collected in these logs, it is essential. Administrators should closely monitor the event logs of every business server and ensure that workstations are configured to track important system events. On servers, you want to ensure that systems are secure, that applications and services are operating normally, and that the server isn’t experiencing errors that could hamper performance. On workstations, you want to ensure that the events you need to maintain systems and resolve problems are being logged, and that the logs are accessible to you as necessary.

The Windows service that manages event logging is called the Windows Event Log service. When this service is started, Windows logs important information. The logs available on a system depend on the system's role and the services installed. Two general types of log files are used:

- **Windows Logs** Logs that the operating system uses to record general system events related to applications, security, setup, and system components
- **Applications And Services Logs** Logs that specific applications and services use to record application-specific or service-specific events

Logs you may see include the following:

- **Application** This log records significant incidents associated with specific applications. For example, Exchange Server logs events related to mail exchange, including events for the information store, mailboxes, and service states. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\Application.Evtx.
- **Directory Service** On domain controllers, this log records incidents from Active Directory Domain Service (AD DS), including events related to directory startup, global catalogs, and integrity checking. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\Directory Service.Evtx.
- **DNS Server** On DNS servers, this log records DNS queries, responses, and other DNS activities. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\DNS Server.Evtx.
- **DFS Replication** On domain controllers using DFS replication, this log records file replication activities on the system, including events for service status and control, scanning data in system volumes, and managing replication sets. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\DFS

Replication.Evtx.

- **File Replication Service** This log records file replication activities on the system. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\File Replication Service.Evtx.
- **Forwarded Events** When event forwarding is configured, this log records forwarded events from other servers. The default location is %SystemRoot%\System32\Winevt\Logs\FordwardedEvents.Evtx.
- **Hardware Events** When hardware subsystem event reporting is configured, this log records hardware events reported to the operating system. The default location is %SystemRoot%\System32\Winevt\Logs\HardwareEvent.Evtx.
- **Microsoft\Windows** A group of logs that track events related to specific Windows services and features. Logs are organized by component type and event category.
- **Security** This log records events related to security such as logon/logoff, privilege use, and resource access. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\Security.Evtx.

NOTE To gain access to security logs, users must be granted the user right Manage Auditing And Security Log. By default, members of the administrators group have this user right.

- **Setup** This log records events logged by the operating system or its components during setup and installation. The default location is: %SystemRoot%\System32\Winevt\Logs\Setup.Evtx.
- **System** This log records events from the operating system or its components, such as the failure of a service to start, driver initialization, system-wide messages, and other messages that relate to the system in general. By default, this log is stored in %SystemRoot%\System32\Winevt\Logs\System.Evtx.

- **Windows PowerShell** This log records activities related to the use of the Windows PowerShell. The default location is %SystemRoot%\System32\Winevt\Logs\Windows PowerShell.Evtx.

Events range in severity from informational messages to general warnings to serious incidents such as critical errors and failures. The category of an event is indicated by its event level. Event levels include

- **Information** Indicates an informational event has occurred, which is generally related to a successful action.
- **Warning** Indicates a general warning. Warnings are often useful in preventing future system problems.
- **Error** Indicates a critical error, such as the failure of a service to start.
- **Audit Success** Indicates the successful execution of an action that you are tracking through auditing, such as privilege use.
- **Audit Failure** Indicates the failed execution of an action that you are tracking through auditing, such as failure to log on.

NOTE Of the many event types, the two you'll want to monitor closely are warnings and errors. Whenever these types of events occur and you're unsure of the reason, you should take a closer look to determine whether you need to take further action.

In addition to level, each event has the following common properties associated with it:

- **Date and Time** Specifies the date and time the event occurred.
- **Source** Identifies the source of the event, such as an application, service, or system component. The event source is useful for pinpointing the cause of an event.
- **Event ID** Details the specific event that occurred with a numeric identifier. Event IDs are generated by the event source and used

to uniquely identify the event.

- **Task Category** Specifies the category of the event, which is sometimes used to further describe the related action. Each event source has its own event categories. For example, with the security source, categories include logon/logoff, privilege use, policy change, and account management.
- **User** Identifies the user account that caused the event to be generated. Users can include special identities, such as Local Service, Network Service, and Anonymous Logon, as well as actual user accounts. The user account can also be listed as N/A to indicate that a user account is not applicable in this situation.
- **Computer** Identifies the computer that caused the event to occur.
- **Description** Provides a detailed description of the event and may also include details about where to find more information to resolve or handle an issue. This field is available when you double-click a log entry in Event Viewer.
- **Data** Any data or error code output by the event

The graphical tool you use to manage events is Event Viewer. You can start this tool by typing **eventvwr** at the command line for the local computer, or **eventvwr *ComputerName***, where *ComputerName* is the name of the remote computer whose events you wish to examine. As with most GUI tools, Event Viewer is easy to use and you will want to continue to use it for certain management tasks. For example, you must use Event Viewer to control the size of the event logs, to specify how logging is handled, and to archive event logs. You cannot perform these tasks at the command line.

Windows 8.1 Windows Server 2012, and Windows Server 2012 R2 provide several different tools and techniques for working with the event logs at the command line, including the following:

- **Powershell Get-Eventlog** Searches classic event logs and collects event entries that match specific criteria. In a script, you could use Powershell Get-Eventlog to examine events in multiple logs and then store the results in a file, making it easier to track information as well as warnings and errors.

- **Eventcreate** Creates custom events in the event logs. Whenever you run custom scripts on a schedule or as part of routine maintenance, you may want to record the action in the event logs and Eventcreate provides a way to do this.
- **Custom views** Uses XPath queries to create custom or filtered views of event logs, allowing you to quickly and easily find events that match specific criteria. Because XPath queries can be used on any compatible system, you can re-create custom and filtered views on other computers simply by running the query on a target computer.

REAL WORLD Monitoring system events isn't something you should do haphazardly. Rather, it is something you should do routinely and thoroughly. With servers, you will want to examine event logs at least once a day. With workstations, you will want to examine logs on specific workstations as necessary, such as when a user reports a problem.

Viewing and Filtering Event Logs

Using the Windows PowerShell Get-Eventlog cmdlet, you can obtain detailed information from the classic event logs. When working with Get-Eventlog, don't overlook the power of automation. You don't have to run the command manually each time from a Windows PowerShell prompt. Instead, you can create a script to query the event logs and then save the results to a file. If you copy that file to a published folder on an intranet server, you can use your Web browser to examine event listings. Not only will that save you time, it will also give you a single location for examining event logs and determining whether any issues require further study.

NOTE In this book, I discuss tools that provide the best way to get the job done using the command line. In this instance, the best way to extract information from the event logs at the command line is to use Windows PowerShell. Although I introduce Windows PowerShell and discuss key cmdlets in this book, this book does not provide in-depth information on Windows PowerShell. For in-depth information on Windows PowerShell, I recommend *Windows PowerShell: The Personal Trainer*.

Viewing Events

You run Get-Eventlog at a Windows PowerShell prompt. The basic syntax for Get-Eventlog is `get-eventlog "LogName"`

where *LogName* is the name of the log you want to work with, such as "Application," "System," "Security," or "Directory Service." In this example, you examine the Application log: `get-eventlog "Application"`

NOTE Technically, the quotation marks are necessary only when the log name contains a space, as is the case with the DNS Server, Directory Service, and File Replication Service logs. However, I recommend using the quotation marks all the time. That way, you won't forget them when they are needed and they won't cause your scripts or scheduled tasks to fail.

TIP To work with the security log, you must use an elevated, administrator prompt.

The output of this query would look similar to the following:

```
Index
Time          Type Source                EventID Message
----- ----          ---- ------                ------- -------
15959 Mar 20 16:56  Erro MSExchange System...     4001 A transient
failure has occurred. The problem may resolve its...

15958 Mar 20 16:55  Erro MSExchange System...     4001 A transient
failure has occurred. The problem may resolve its...

15957 Mar 20 16:54  Erro MSExchange System...     4001 A transient
failure has occurred. The problem may resolve its...
```

```
15956 Mar 20 16:53  Erro MSExchange System...     4001 A transient
failure has occurred. The problem may resolve its...
```

As you can see, the output shows the Index, Time, Type, Source, Event ID, and Message properties of events. Because the index is the position of the event in the log, this example lists events 15,956 to 15,959. When you follow Get-Eventlog with the log name, the –Logname parameter is implied. You can also specify the –Logname parameter directly as shown in this example: `get-eventlog –logname "security"`

By default, Get-Eventlog returns every event in the specified event log from newest to oldest. In most cases, this is simply too much information and you'll need to filter the events to get a usable amount of data. One way to filter the event logs is to specify that you only want to see details about the newest events. For example, you might want to see only the 100 newest events in a log.

Using the –Newest parameter, you can limit the return to the newest events. The following example lists the 100 newest events in the security log: `get-eventlog "security" -newest 50`

Unlike previous command-line utilities that we've worked with, Get-Eventlog is a Windows PowerShell cmdlet. If this is your first time working with Windows PowerShell, you'll need to ensure that the feature is installed on your computer. If you don't want to invoke a separate Windows PowerShell instance, you can invoke Windows PowerShell only to run the Get-Eventlog cmdlet, as shown in this example: `powershell.exe get-eventlog –logname "security"`

You also could insert this command in a batch script. In a batch script, this command would invoke Windows PowerShell, execute the Get-Eventlog cmdlet, and then exit Windows PowerShell.

Filtering Events

One of the key reasons for using Get-Eventlog is its ability to group and filter events in the result set. When you group events by type, you can more easily separate informational events from critical, warning, and error events. When you group by source, you can more easily track events from specific sources. When you group by event ID, you can more easily correlate the recurrence of specific events.

You can group events by source, eventid, entrytype, and timegenerated using the following technique: 1. Get the events you want to work with and store them in the $e variable by entering: `$e = get-eventlog -newest 100 -logname "application"`

2. Use the Group-Object cmdlet to group the event objects stored in $e by a specified property. In this example, you group by eventid: `$e | group-object -property eventide` Another way to work with events is to sort them according to a specific property. You can sort by source, eventid, entrytype, or timegenerated using the following technique: 1. Get the events you want to work with and store them in the *$e* variable by entering: `$e = get-eventlog -newest 100 -logname "application"`

2. Use the Sort-Object cmdlet to sort the event objects stored in *$e* by a specified property. In this example, you sort by entrytype: `$e | sort-object -property entrytype` Typically, you won't want to see every event generated on a system. More often, you will want to see only warnings or critical errors, and that is precisely what filters are for. Using filters, you can include only events that match the criteria you specify. To do this, you would search the EntryType property for occurrences of the word *error*. Here is an example: 1. Get the events you want to work with and store them in the *$e* variable by entering: `$e = get-eventlog -newest 500 -logname "application"`

2. Use the Where-Object cmdlet to search for specific text in a named property of the event objects stored in *$e*. In this example, you match events with the error entry type: `$e | where-object {$_.EntryType -match "error"}`

The Where-Object cmdlet uses a search algorithm that is not case-sensitive, meaning you could enter Error, error, or ERROR to match error events. You can also search for warning, critical, and information events. Because Where-Object considers partial text matches to be valid, you don't want to enter the full entry type. You could also search for warn,

crit, or info, such as: `$e = get-eventlog -newest 100 -logname "application"`
```
$e | where-object {$_.EntryType -match "warn"}
```

You can use Where-Object with other event object properties as well. The following example searches for event sources containing the text MSDTC:
```
$e = get-eventlog -newest 500 -logname "application"
$e | where-object {$_.Source -match "MSDTC}
```

The following example searches for event ID 15001: `$e = get-eventlog -newest 500 -logname "application"`
```
$e | where-object {$_.EventID -match "15001"}
```

You can automate the event querying process by creating a Windows PowerShell script that obtains the event information you want to see and then writes it to a text file. Consider the following example: `$e = get-eventlog -newest 100 -logname "system"`
```
$e | where-object {$_.EntryType -match "error"} > clog.txt

$e = get-eventlog -newest 500 -logname "application"
$e | where-object {$_.EntryType -match "error"} >> clog.txt

$e = get-eventlog -newest 500 -logname "directory service"
$e | where-object {$_.EntryType -match "error"} >> clog.txt
```
Here, you are examining the system, application and directory service event logs and writing any resulting output to a network share on CorpIntranet01. If any of the named logs have error events among the 500 most recent events in the logs, the errors are written to the Currentlog.txt file. Because the first redirection is overwrite (>) and the remaining entries are append (>>), any existing Clog.txt file is overwritten each time the script runs. This ensures that only current events are listed. To take the automation process a step further, you can create a scheduled task that runs the script each day or at specific intervals during the day.

Windows PowerShell script files have the .ps1 filename extension. (Note that this is the letter P, the letter S, and the digit one.) To run a script at the Windows PowerShell prompt, you type the name of the script and optionally, the filename extension. You must specify the full qualified path to the script file, even if the script is in the current directory. To indicate the current directory, type the directory name or use the dot (.) to represent the current directory. Following this, if you saved the

Windows PowerShell script as a file called CheckEvents.ps1 in the current directory, you could run the script by entering .\checkevents.ps1 at the Windows PowerShell prompt.

Writing Custom Events to the Event Logs

Whenever you work with automated scripts, scheduled tasks, or custom applications, you might want those scripts, tasks, or applications to write custom events to the event logs. For example, if a script runs normally, you might want to write an informational event in the application log that specifies this so it is easier to determine that the script ran and completed normally. Similarly, if a script doesn't run normally and generates errors, you might want to log an error or warning event in the application log so that you'll know to examine the script and determine what happened.

> *TIP* You can track errors that occur in scripts using *%ErrorLevel%*. This environment variable tracks the exit code of the most recently used command. If the command executes normally, the error level is zero (0). If an error occurs while executing the command, the error level is set to a nonzero value. To learn moreabout working with error levels, see the section titled "Getting Acquainted with Variables" in Chapter 3, "Command-Line Scripting Essentials."

To create custom events, you'll use the Eventcreate utility. Custom events can be logged in any available log except the security log, and can include the event source, ID, and description you want to use. The syntax for Eventcreate is `eventcreate /l` *LogName* `/so` *EventSource* `/t` *EventType* `/id EventID`
`/d` *EventDescr* where

- **LogName** Sets the name of the log to which the event should be written. Use quotation marks if the log name contains spaces, as in "DNS Server."

> *TIP* You cannot write custom events to the security log. You can, however, write custom events to the other logs. Start by writing a

dummy event using the event source you want to register for use with that log. The initial event for that source will be written to the application log. You can then use the source with the specified log and your custom events.

- **EventSource** Specifies the source to use for the event and can be any string of characters. If the string contains spaces, use quotation marks, as in “Event Tracker.” In most cases, you’ll want the event source to identify the application, task, or script that is generating the error.

CAUTION Carefully plan the event source you want to use before you write events to the logs using those sources. Each event source you use must be unique and cannot have the same name as an existing source used by an installed service or application. Further, you shouldn’t use event source names used by Windows roles, role services, or features. For example, you shouldn’t use DNS, W32Time, or Ntfrs as sources because these sources are used by Windows Server 2012 and Windows Server 2012 R2. Additionally, once you use an event source with a particular log, the event source is registered for use with that log on the specified system. For example, you cannot use “EventChecker” as a source in the application log and in the system log on MAILER1. If you try to write an event using “EventChecker” to the system log after writing a previous event with that source to the application log, you will see the following error message: “ERROR: Source already exists in 'Application' log. Source cannot be duplicated.”

- **EventType** Sets the event type as Information, Warning, or Error. “Success Audit” and “Failure Audit” event types are not valid; these events are used with the security logs and you cannot write custom events to the security logs.
- **EventID** Specifies the numeric ID for the event and can be any value from 1 to 1,000. Before you assign event IDs haphazardly, you may want to create a list of the general events that can

occur and then break these down into categories. You could then assign a range of event IDs to each category. For example, events in the 100s could be general events, events in the 200s could be status events, events in the 500s could be warning events, and events in the 900s could be error events.

- **EventDescr** Sets the description for the event and can be any string of characters. Be sure to enclose the description in quotation marks.

NOTE Eventcreate runs by default on the local computer with the permissions of the user who is currently logged on. As necessary, you can also specify the remote computer whose tasks you want to query and the Run As permissions using /S *Computer* /u [*Domain*\]User [/P *Password*], where *Computer* is the remote computer name or IP address, *Domain* is the optional domain name in which the user account is located, User is the name of the user account whose permissions you want to use, and *Password* is the optional password for the user account.

To see how you can use Eventcreate, consider the following examples: Create an information event in the application log with the source Event Tracker and event ID 209: `eventcreate l "application" t information /so`
```
"Event Tracker"
/id 209 /d "evs.bat script ran without errors."
```

Create a warning event in the system log with the source CustApp and event ID 511: `eventcreate /l "system" t warning so "CustApp" id 511 d`
```
"sysck.exe didn't complete successfully."
```

Create an error event in the system log on MAILER1 with the source “SysMon” and event ID 918: `eventcreate s Mailer1 l "system" t error so`
```
"SysMon" /id 918
/d "sysmon.exe was unable to verify write operation."
```

Creating and Using Saved Queries

Event Viewer supports XPath queries for creating custom views and

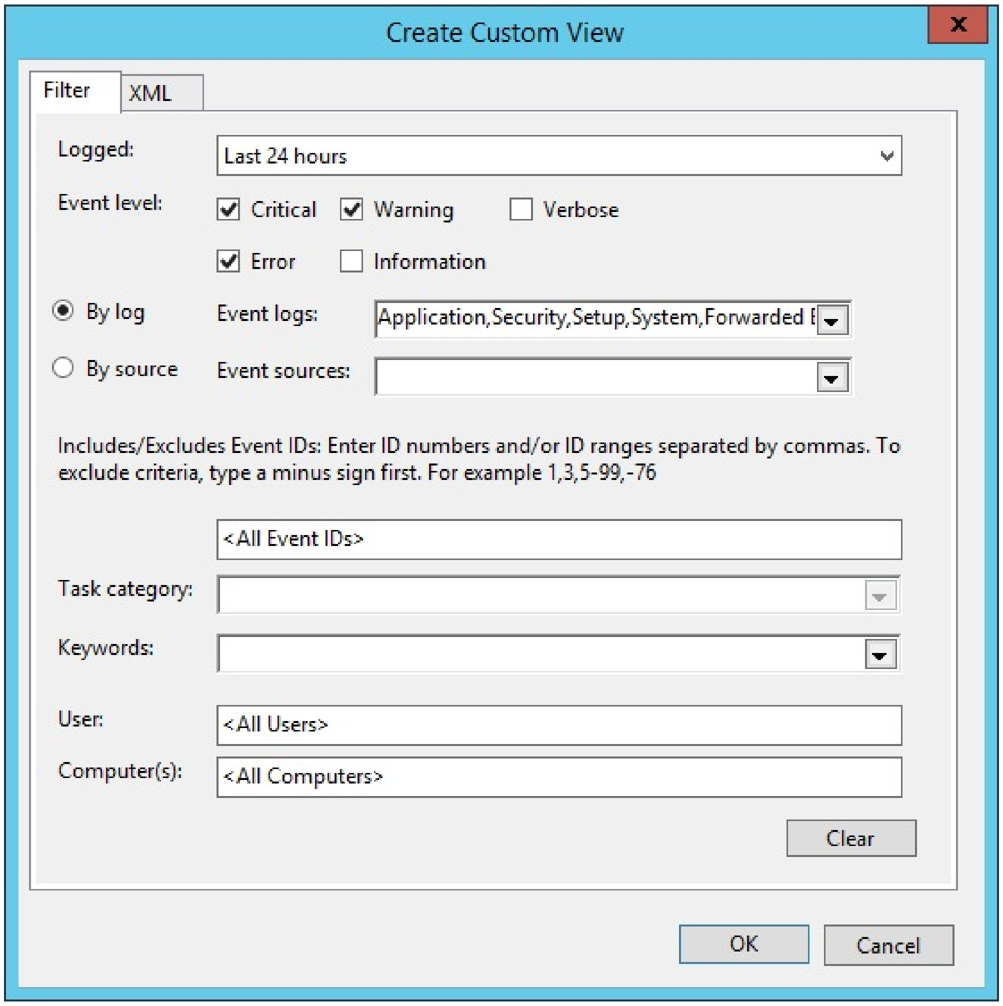
Event Viewer supports XPath queries for creating custom views and filtering event logs. XPath is a non-XML language used to identify specific parts of XML documents. Event Viewer uses XPath expressions that match and select elements in a source log and copy them to a destination log to create a custom or filtered view.

When you are creating a custom or filtered view in Event Viewer, you can copy the XPath query and save it to an Event Viewer Custom View file. By running this query again, you can re-create the custom view or filter on any computer running Windows 8.1, Windows Server 2012, or Windows Server 2012 R2. For example, if you create a filtered view of the application log that helps you identify a problem with Microsoft SQL Server, you could save the related XPath query to a Custom View file so that you can create the filtered view on other computers in your organization.

Event Viewer creates several filtered views of the event logs for you automatically. Filtered views are listed under the Custom Views node. When you select the Administrative Events node, you'll see a list of all errors and warnings for all logs. When you expand the Server Roles node and then select a role-specific view, you'll see a list of all events for the selected role.

You can create and save your own custom view by following these steps:

1. Start Event Viewer by opening Server Manager, clicking Tools, and then clicking Event Viewer.
2. Select the Custom Views node. In the Actions pane or on the Action menu, click Create Custom View.
3. In the Create Custom View dialog box, shown in Figure 6-1, use the Logged list to select the included time frame for logged events. You can choose to include events from Anytime, the Last Hour, Last 12 Hours, Last 24 Hours, Last 7 Days, or Last 30 Days.


Create Custom View

Filter | XML

Logged: Last 24 hours

Event level: [✓ Critical] [✓ Warning] [ ] Verbose [✓ Error] [ ] Information

[● By log] Event logs: Application,Security,Setup,System,Forwarded E

[ ] By source Event sources:

Includes/Excludes Event IDs: Enter ID numbers and/or ID ranges separated by commas. To exclude criteria, type a minus sign first. For example 1,3,5-99,-76

<All Event IDs>

Task category:

Keywords:

User: <All Users>

Computer(s): <All Computers>

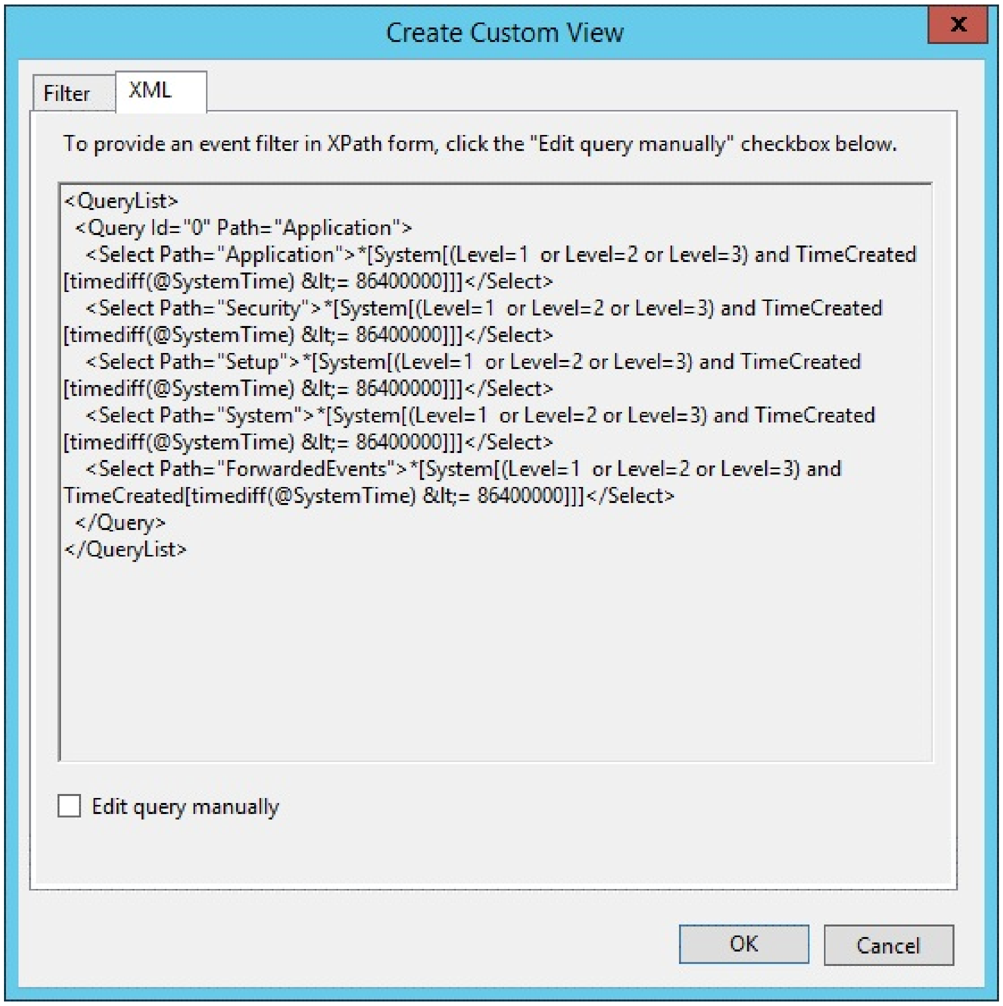
FIGURE 6-1 Create a filter to specify the types of events to display.

4. Use the Event Level check boxes to specify the level of events to include. Select Verbose to get additional detail.
5. You can create a custom view for either a specific set of logs or a specific set of event sources: Use the Event Logs list to select event logs to include. You can select multiple event logs by selecting their related check boxes. If you select specific event logs, all other event logs are excluded.

   Use the Event Sources list to select event sources to include. You can select multiple event sources by selecting their related check boxes. If you select specific event sources, all other event sources are excluded.
6. Optionally, use the User and Computer(s) boxes to specify users and

computers that should be included. If you do not specify the users and computers to be included, events generated by all users and computers are included.

7. Click the XML tab to display the related XPath query, as shown in Figure 6-2.



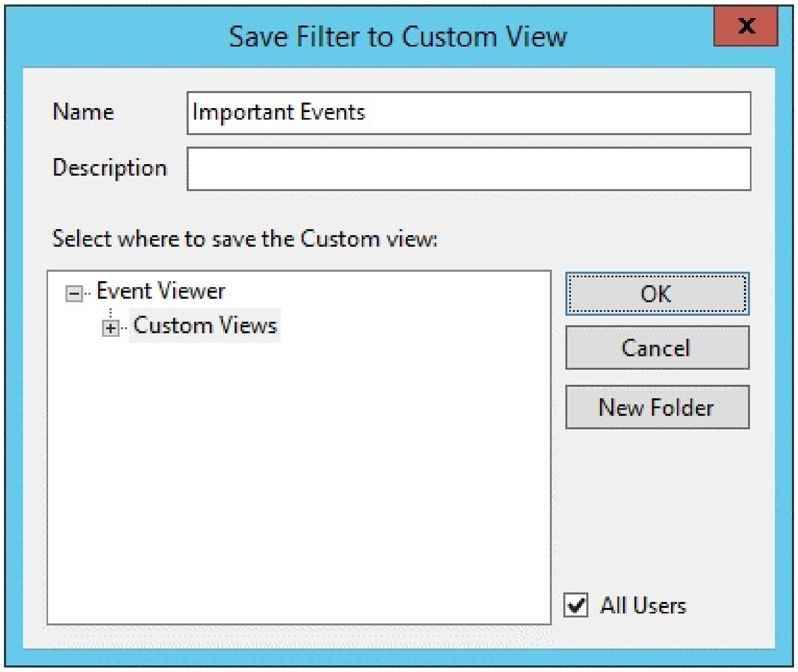
To provide an event filter in XPath form, click the "Edit query manually" checkbox below.

```
<QueryList>
  <Query Id="0" Path="Application">
    <Select Path="Application">*[System[(Level=1  or Level=2 or Level=3) and TimeCreated
[timediff(@SystemTime) &lt;= 86400000]]]</Select>
    <Select Path="Security">*[System[(Level=1  or Level=2 or Level=3) and TimeCreated
[timediff(@SystemTime) &lt;= 86400000]]]</Select>
    <Select Path="Setup">*[System[(Level=1  or Level=2 or Level=3) and TimeCreated
[timediff(@SystemTime) &lt;= 86400000]]]</Select>
    <Select Path="System">*[System[(Level=1  or Level=2 or Level=3) and TimeCreated
[timediff(@SystemTime) &lt;= 86400000]]]</Select>
    <Select Path="ForwardedEvents">*[System[(Level=1  or Level=2 or Level=3) and
TimeCreated[timediff(@SystemTime) &lt;= 86400000]]]</Select>
  </Query>
</QueryList>
```

☐ Edit query manually

FIGURE 6-2 Review the related XPath query.

8. Click OK to close the Create Custom View dialog box. In the Save Filter To Custom View dialog box, shown in Figure 6-3, type a name and description for the custom view.


Save Filter to Custom View

Name: Important Events

Description:

Select where to save the Custom view:

- Event Viewer
  - Custom Views

FIGURE 6-3 Save the filtered view.

9. Select where to save the custom view. By default, custom views are saved under the Custom View node. You can create a new node by clicking New Folder, entering the name of the new folder, and then clicking OK.
10. Click OK to close the Save Filter To Custom View dialog box. You should now see a filtered list of events.
11. Right-click the custom view and then select Export Custom View. Use the Save As dialog box to select a save location and enter a filename for the Event Viewer Custom View file.

The Custom View file contains the XPath query that was displayed on the XML tab previously. Members of the Event Log Readers group, administrators, and others with appropriate permissions can run the query to view events on remote computers using the following syntax: `eventvwr ComputerName /v: QueryFile` where *ComputerName* is the name of the remote computer whose events you wish to examine and *QueryFile* is the name or full path to the Custom View file containing the XPath query, such as: `eventvwr mailserver25 /v: importantevents.xml` When Event Viewer starts, you'll find the custom view under the Custom Views node.

Monitoring Performance: The Essentials

Now that you know how to view, filter, and create events, let’s look at techniques that you can use to monitor a computer’s performance. Windows 8.1 Windows Server 2012, and Windows Server 2012 R2 have several tools for this purpose and in this section, we’ll look at the Typeperf command-line utility. In the next chapter, you’ll learn about other tools for monitoring performance.

Understanding Performance Monitoring at the Command Line

Typeperf is a tool designed to track and display performance information in real time. It gathers information on any performance parameters you’ve configured for monitoring and presents it on the command line. Each performance item you want to monitor is defined by the following three components:

- **Performance object** Represents any system component that has a set of measurable properties. A performance object can be a physical part of the operating system, such as the memory, the processor, or the paging file; a logical component, such as a logical disk or print queue; or a software element, such as a process or a thread.
- **Performance object instance** Represents single occurrences of performance objects. If a particular object has multiple instances, such as when a computer has multiple processors or multiple disk drives, you can use an object instance to track a specific occurrence of that object. You could also elect to track all instances of an object, such as whether you want to monitor all processors on a system.
- **Performance counter** Represents measurable properties of performance objects. For example, with a paging file, you can measure the percentage utilization using the %Usage counter.

[illegible]

In a standard installation of Windows 8.1, Windows Server 2012, or Windows Server 2012 R2, many performance objects are available for monitoring. As you add services, applications, and components, additional performance objects can become available. For example, when you install the Domain Name System (DNS) on a server, the DNS object becomes available for monitoring on that computer.

The most common performance objects you'll want to monitor are summarized in Table 6-1. Like all performance objects, each performance object listed here has a set of counters that you can track. The most commonly tracked performance objects are Memory, PhysicalDisk, and Processor.

TABLE 6-1 Commonly Tracked Performance Objects **TABLE 6-1** Commonly Tracked Performance Objects

PERFORMANCE OBJECT	DESCRIPTION
Cache	Monitors the file system cache, which is an area of physical memory that indicates application I/O activity.
Database ==> Instances	Monitors performance for instances of the embedded database management system used by the operating system.
IPv4	Monitors IPv4 communications and related activities.
IPv6	Monitors IPv6 communications and related activities.
LogicalDisk	Monitors the logical volumes on a computer.
Memory	Monitors memory performance for system cache (including pooled, paged memory and pooled, nonpaged memory), physical memory, and virtual memory.
Network Interface	Monitors the network adapters configured on the computer.

Objects

Monitors the number of events, mutexes, processes, sections, semaphores, and threads on the computer.

Paging File

Monitors page file current and peak usage.

PhysicalDisk

Monitors hard disk read/write activity as well as data transfers, hard faults, and soft faults.

Print Queue

Monitors print jobs, spooling, and print queue activity.

Process

Monitors all processes running on a computer.

Processor

Monitors processor idle time, idle states, usage, deferred procedure calls, and interrupts.

Server

Monitors communication between the computer and the network as well as important usage statistics, including logon errors, access errors, and user sessions.

Server Work Queues Monitors threading and client requests.

System

Monitors system-level counters, including processes, threads, context switching of threads, file system control operations, system calls, and system uptime.

TCPv4

Monitors TCPv4 communications and related activities.

TCPv6

Monitors TCPv6 communications and related activities.

Thread

Monitors all running threads and allows you to examine usage statistics for individual threads by process ID.

UDPv4	Monitors UDPv4 communications and related activities.
UDPv6	Monitors UDPv6 communications and related activities.

Tracking Performance Data

Using Typeperf, you can write performance data to the command line or to a log file. The key to using Typeperf is to identify the pathnames of the performance counters you want to track. The performance counter path has the following syntax: `\\ComputerName\ObjectName\ObjectCounter` where *ComputerName* is the computer name or IP address of the local or remote computer you want to work with, *ObjectName* is the name of a counter object, and *ObjectCounter* is the name of the object counter to use. For example, if you wanted to track the available memory on FileServer42, you'd type the following: `typeperf "\\fileserver42\memory\available mbytes"`

> *NOTE* Enclosing the counter path in double quotation marks is required in this example because the counter path includes spaces. Although double quotation marks aren't always required, it is good form to always use them.

You can also easily track all counters for an object by using an asterisk (*) as the counter name, such as in the following example: `typeperf "\\fileserver42\Memory\*"`

Here, you track all counters for the Memory object.

When objects have multiple instances, such as the Processor or LogicalDisk object, you must specify the object instance you want to work with. The syntax for this is as follows: `\\ComputerName\ObjectName(ObjectInstance)\ObjectCounter` Here, you follow the object name with the object instance in parentheses. When an object has multiple instances, you can work with all instances of that object using _Total as the instance name. You can work with a specific instance of an object by using its instance identifier. For example, if you want to examine the Processor\%Processor Time counter, you can use this command line to work with all processor instances: `typeperf "\\fileserver42\Processor(_Total)\%Processor Time"`

or this command line to work with a specific processor instance: `typeperf "\\fileserver42\Processor(0)\%Processor Time"`

Here, Processor(0) identifies the first processor on the system.

Typeperf has many available parameters, as summarized in Table 6-2.

TABLE 6-2 Parameters for Typeperf

PARAMETER	**DESCRIPTION**
–cf <*filename*>	Identifies a file containing a list of performance counters to monitor.
–config <*filename*>	Identifies a settings file containing command options.
–f <CSV|TSV|BIN|SQL>	Sets the output file format. The default is .csv for comma-separated values.
–o <*filename*>	Sets the path of an output file or SQL database.
–q [*object*]	Lists installed counters for the specified object.
–qx [*object*]	Lists installed counters with instances.
–s <*ComputerName*>	Sets the remote computer to monitor if no computer is specified in the counter path.
–sc <*samples*>	Sets the number of samples to collect.
–si <[[hh:]*mm*:]ss>	Sets the time between samples. The default is 1 second.
–y	Answers Yes to all questions without prompting.

Typeperf writes its output to the command line in a comma-delimited list by default. You can redirect the output to a file using the –O parameter and set the output format using the –F parameter. The output format indicators are CSV for a comma-delimited text file, TSV for a tab-delimited text file, BIN for a binary file, and SQL for an SQL binary file. Consider the following example: `typeperf "\\fileserver42\Memory\*" -o memperf.bin -f bin` Here, you track all counters for the Memory object and write the output to a binary file called MemPerf.bin in the current directory.

If you need help determining the available counters for an object, type **typeperf –q** followed by the object name for which you want to view counters. For example, if you enter the following command: `typeperf -q Memory` you'll see a list of available counters similar to the following:

```
\memory\Page Faults/sec
\memory\Available Bytes
\memory\Committed Bytes
\memory\Commit Limit
\memory\Write Copies/sec
\memory\Transition Faults/sec
\memory\Cache Faults/sec
\memory\Demand Zero Faults/sec
\memory\Pages/sec
\memory\Pages Input/sec
\memory\Page Reads/sec
\memory\Pages Output/sec
\memory\Pool Paged Bytes
\memory\Pool Nonpaged Bytes
\memory\Page Writes/sec
\memory\Pool Paged Allocs
\memory\Pool Nonpaged Allocs
\memory\Free System Page Table Entries
\memory\Cache Bytes
\memory\Cache Bytes Peak
\memory\Pool Paged Resident Bytes
\memory\System Code Total Bytes
\memory\System Code Resident Bytes
\memory\System Driver Total Bytes
\memory\System Driver Resident Bytes
\memory\System Cache Resident Bytes
\memory\% Committed Bytes In Use
\memory\Available KBytes
```

```
\memory\Available MBytes
\memory\Transition Pages RePurposed/sec
\memory\Free & Zero Page List Bytes
\memory\Modified Page List Bytes
\memory\Standby Cache Reserve Bytes
\memory\Standby Cache Normal Priority Bytes
\memory\Standby Cache Core Bytes
```

If an object has multiple instances, you can list the installed counters with instances by using the –Qx parameter, such as in the following: `typeperf -qx PhysicalDisk` The output is a long list of available counters according to their object instances, such as:

```
\PhysicalDisk(0 E: C:)\Current Disk Queue Length
\PhysicalDisk(1 D:)\Current Disk Queue Length
\PhysicalDisk(2 I:)\Current Disk Queue Length
\PhysicalDisk(3 J:)\Current Disk Queue Length
\PhysicalDisk(4 K:)\Current Disk Queue Length
\PhysicalDisk(5 L:)\Current Disk Queue Length
\PhysicalDisk(6 N:)\Current Disk Queue Length
\PhysicalDisk(7 O:)\Current Disk Queue Length
\PhysicalDisk(8 P:)\Current Disk Queue Length
\PhysicalDisk(9 Q:)\Current Disk Queue Length
\PhysicalDisk(_Total)\Current Disk Queue Length
\PhysicalDisk(0 E: C:)\% Disk Time
\PhysicalDisk(1 D:)\% Disk Time
\PhysicalDisk(2 I:)\% Disk Time
\PhysicalDisk(3 J:)\% Disk Time
\PhysicalDisk(4 K:)\% Disk Time
\PhysicalDisk(5 L:)\% Disk Time
\PhysicalDisk(6 N:)\% Disk Time
\PhysicalDisk(7 O:)\% Disk Time
\PhysicalDisk(8 P:)\% Disk Time
\PhysicalDisk(9 Q:)\% Disk Time
\PhysicalDisk(_Total)\% Disk Time
\PhysicalDisk(0 E: C:)\Avg. Disk Queue Length
\PhysicalDisk(1 D:)\Avg. Disk Queue Length
\PhysicalDisk(2 I:)\Avg. Disk Queue Length
\PhysicalDisk(3 J:)\Avg. Disk Queue Length
\PhysicalDisk(4 K:)\Avg. Disk Queue Length
\PhysicalDisk(5 L:)\Avg. Disk Queue Length
\PhysicalDisk(6 N:)\Avg. Disk Queue Length
\PhysicalDisk(7 O:)\Avg. Disk Queue Length
\PhysicalDisk(8 P:)\Avg. Disk Queue Length
\PhysicalDisk(9 Q:)\Avg. Disk Queue Length
...
\PhysicalDisk(0 E: C:)\% Idle Time
\PhysicalDisk(1 D:)\% Idle Time
\PhysicalDisk(2 I:)\% Idle Time
\PhysicalDisk(3 J:)\% Idle Time
\PhysicalDisk(4 K:)\% Idle Time
```

```
\PhysicalDisk(5 L:)\% Idle Time
\PhysicalDisk(6 N:)\% Idle Time
\PhysicalDisk(7 O:)\% Idle Time
\PhysicalDisk(8 P:)\% Idle Time
\PhysicalDisk(9 Q:)\% Idle Time
\PhysicalDisk(_Total)\% Idle Time
\PhysicalDisk(0 E: C:)\Split IO/Sec
\PhysicalDisk(1 D:)\Split IO/Sec
\PhysicalDisk(2 I:)\Split IO/Sec
\PhysicalDisk(3 J:)\Split IO/Sec
\PhysicalDisk(4 K:)\Split IO/Sec
\PhysicalDisk(5 L:)\Split IO/Sec
\PhysicalDisk(6 N:)\Split IO/Sec
\PhysicalDisk(7 O:)\Split IO/Sec
\PhysicalDisk(8 P:)\Split IO/Sec
\PhysicalDisk(9 Q:)\Split IO/Sec
\PhysicalDisk(_Total)\Split IO/Sec
```

You can use this counter information as input to Typeperf as well. Add the –O parameter and write the output to a text file, such as in the following: `typeperf -qx PhysicalDisk -o perf.txt` Then edit the text file so that only the counters you want to track are included. You can then use the file to determine which performance counters are tracked by specifying the –Cf parameter followed by the file path to this counter file. Consider the following example: `typeperf -cf perf.txt -o c:\perflogs\perf.bin -f bin` Here, Typeperf reads the list of counters to track from Perf.txt and then writes the performance data in binary format to a file in the C:\perflogs directory.

By default, Typeperf samples data once every second until you tell it to stop by pressing Ctrl+C. This may be okay when you are working at the command line and actively monitoring the output. However, it doesn’t work so well when you have other things to do and can’t actively monitor the output—which is probably most of the time. Therefore, you’ll usually want to control the sampling interval and duration.

You can use the –Si and –Sc parameters to control the sampling interval and set how long to sample, respectively. For example, if you wanted Typeperf to sample every 120 seconds and stop logging after 100 samples, you could type this: `typeperf -cf perf.txt -o c:\perflogs\perf.bin -f bin -si 120 -sc 100`

Chapter 8. Managing Processes and Maintaining Performance

It can be extremely frustrating when programs won't run or respond. Often you'll find that the source of the problem is a particular process or program. Sometimes, though you need to dig deeper to search out the cause of the performance problem and correct it. Hopefully, by pinpointing the cause of a problem, you can prevent it from happening again.

Managing Applications, Processes, and Performance

Whenever the operating system or a user starts a service, runs an application, or executes a command, Windows starts one or more processes to handle the related program. Several command-line utilities are available to help you manage and monitor programs. These utilities include the following:

- **Task List (Tasklist)** Lists all running processes by name and process ID. Includes information on the user session and memory usage.
- **Task Kill (Taskkill)** Stops running processes by name or process ID. Using filters, you can also halt processes by process status, session number, CPU time, memory usage, user name, and more.
- **Powershell get-process** Displays performance statistics, including memory and CPU usage, as well as a list of all running processes. Used to get a detailed snapshot of resource usage and running processes. Available when you install Windows PowerShell.

In the sections that follow, you'll find detailed discussions on how these command-line tools are used. First, however, let's look at the ways processes are run and the common problems you may encounter when working with them.

Understanding System and User Processes

Generally, processes that the operating system starts are referred to as *system processes*; processes that users start are referred to as *user processes*. Most user processes are run in interactive mode. That is, a user starts a process interactively with the keyboard or mouse. If the application or program is active and selected, the related interactive process has control over the keyboard and mouse until you switch control by terminating the program or selecting a different one. When a process has control, it's said to be running "in the foreground."

Processes can also run in the background, independently of user logon sessions. Background processes do not have control over the keyboard, mouse, or other input devices and are usually run by the operating system. Using the Task Scheduler, users can run processes in the background as well, however, and these processes can operate regardless of whether the user is logged on. For example, if Task Scheduler starts a scheduled task while the user is logged on, the process can continue even when the user logs off.

Windows tracks every process running on a system by image name, process ID, priority, and other parameters that record resource usage. The image name is the name of the executable that started the process, such as Msdtc.exe or Svchost.exe. The process ID is a numeric identifier for the process, such as 2588. The process ID is a priority-indicator of how much of the system's resources the process should get relative to other running processes. With priority processing, a process with a higher priority gets preference over processes with lower priority and may not have to wait to get processing time, access memory, or work with the file system. A process with lower priority, on the other hand, usually must wait for a higher-priority process to complete its current task before gaining access to the CPU, memory, or the file system.

In a perfect world, processes would run perfectly and would never have problems. The reality is, however, that problems occur and they often appear when you'd least want them to. Common problems include the following:

- Processes become nonresponsive, such as when an application stops processing requests. When this happens, users may tell you that they can’t access a particular application, that their requests aren’t being handled, or that they were kicked out of the application.
- Processes fail to release the CPU, such as when you have a runaway process that is using up CPU time. When this happens, the system may appear to be slow or nonresponsive because the runaway process is hogging processor time and is not allowing other processes to complete their tasks.
- Processes use more memory than they should, such as when an application has a memory leak. When this happens, processes aren’t properly releasing memory that they’re using. As a result, the system’s available memory may gradually decrease over time and as the available memory gets low, the system may be slow to respond to requests or it may become nonresponsive. Memory leaks can also make other programs running on the same system behave erratically.

In most cases, when you detect these or other problems with system processes, you’ll want to stop the process and start it again. You would also want to examine the event logs to see whether you can determine the cause of the problem. In the case of memory leaks, you want to report the memory leak to the developers and see whether an update that resolves the problem is available.

A periodic restart of an application with a known memory leak is often useful. Restarting the application should allow the operating system to recover any lost memory.

Examining Running Processes

When you want to examine processes that are running on a local or remote system, you can use the Tasklist command-line utility. With Tasklist, you can do the following: Obtain the process ID, status, and other important information about processes running on a system.

- View the relationship between running processes and services configured on a system.
- View lists of DLLs used by processes running on a system.
- Use filters to include or exclude processes from Tasklist queries.

Each of these tasks is discussed in the sections that follow.

Obtaining Detailed Information on Processes

On a local system, you can view a list of running tasks simply by typing **tasklist** at the command prompt. As with many other command-line utilities, Tasklist runs by default with the permissions of the currently logged-on user, and you can also specify the remote computer whose tasks you want to query and the Run As permissions. To do this, use the expanded syntax, which includes the following parameters: `s Computer u [Domain\]User [/p Password]`

where *Computer* is the remote computer name or IP address, *Domain* is the optional domain name in which the user account is located, *User* is the name of the user account whose permissions you want to use, and Password is the optional password for the user account. If you don't specify the domain, the current domain is assumed. If you don't provide the account password, you are prompted for the password.

To see how you can add computer and user information to the syntax, consider the following examples: Query Mailer1 for running tasks:

```
tasklist /s mailer1
```

Query 192.168.1.5 for running tasks using the account adatum\wrstanek: `tasklist /s 192.168.1.5 /u imaginedl\wrstanek` The basic output of these commands is in table format. You can also format the output as a list or lines of comma-separated values using /Fo List or *Fo Csv, respectively. Remember that you can redirect the output to a file using output redirection (> or >>), such as tasklist* s mailer1 >> current-tasks.log.

Regardless of whether you are working with a local or remote computer, the output should be similar to the following:
```
Image Name          PID
Session Name        Session#    Mem Usage
============        ==== ======== ========= ======== ============
System Idle Process  0 Services                  0           28 K
System               4 Services                  0       28,952 K
smss.exe           488 Services                  0          776 K
csrss.exe          560 Services                  0        5,272 K
wininit.exe        608 Services                  0        4,056 K
csrss.exe          620 Console                   1       13,004 K
services.exe       652 Services                  0        7,456 K
lsass.exe          664 Services                  0        1,852 K
lsm.exe            680 Services                  0        6,400 K
svchost.exe        836 Services                  0        7,228 K
winlogon.exe       868 Console                   1        5,544 K
svchost.exe        932 Services                  0        9,440 K
svchost.exe        984 Services                  0       23,304 K
svchost.exe       1048 Services                  0       12,208 K
svchost.exe       1100 Services                  0       71,696 K
svchost.exe       1132 Services                  0       36,920 K
dwm.exe           2832 Console                   1       65,456 K
explorer.exe      2892 Console                   1       25,624 K
```

The Tasklist fields provide the following information:

- **Image Name** The name of the process or executable running the process.
- **PID** The process identification number.
- **Session Name** The name of the session from which the process is being run. An entry of console means the process was started locally.
- **Session #** A numerical identifier for the session.
- **Memory Usage** The total amount of memory being used by the process at the specific moment that Tasklist was run.

If you want more detailed information you can specify that verbose mode should be used by including the /V parameter. Verbose mode adds the following columns of data:

- **Status** Current status of the process as Running, Not Responding, or Unknown. A process can be in an Unknown state and still be running and responding normally. A process that is Not Responding, however, more than likely must be stopped or restarted.
- **User Name** User account under which the process is running, listed in domain\user format. For processes started by Windows, you will see the name of the system account used, such as SYSTEM, LOCAL SERVICE, or NETWORK SERVICE, with the domain listed as NT AUTHORITY.
- **CPU Time** The total amount of CPU-cycle time used by the process since its start.
- **Window Title** Windows display name of the process if available. Otherwise, the display name is listed as N/A for not available. For example, the HelpPane.exe process is listed with the title Windows Help And Support Center

If you use Tasklist to examine running processes, you'll note two unique processes: System and System Idle Process. System shows the resource usage for the local system process. System Idle Process tracks the amount of CPU processing time that isn't being used. Thus, a 99 in the CPU column for the System Idle Process means 99 percent of the system resources currently aren't being used. If you believe that a system is overloaded, you should monitor the idle process. Watch the CPU usage and the total CPU time. If the system consistently has low idle time (meaning high CPU usage), you may want to consider upgrading the processor or even adding processors.

As you examine processes, keep in mind that a single application might start multiple processes. Generally, these processes are dependent on a central process, and from this main process a process tree containing dependent processes is formed. When you terminate processes, you'll usually want to target the main application process or the application itself rather than dependent processes. This ensures that the application is stopped cleanly.

Viewing the Relationship Between Running Processes and Services

When you use Tasklist with the /Svc parameter, you can examine the relationship between running processes and services configured on the system. In the output, you'll see the process image name, process ID, and a list of all services that are using the process, similar to that shown in the following example:

```
Image Name                PID Services
==========                === ============


System Idle Process         0 N/A System                   4 N/A

smss.exe                  288 N/A csrss.exe              404 N/A

csrss.exe                 456 N/A wininit.exe            464 N/A

winlogon.exe              508 N/A services.exe           560 N/A
lsass.exe                 568 Kdc, Netlogon, SamSs
svchost.exe               720 BrokerInfrastructure, DcomLaunch, LSM,

PlugPlay, Power, SystemEventsBroker svchost.exe              772

RpcEptMapper, RpcSs LogonUI.exe                848 N/A

dwm.exe                   860 N/A svchost.exe            896 Dhcp,

EventLog, lmhosts, Wcmsvc TrustedInstaller.exe      956 TrustedInstaller
svchost.exe               996 EventSystem, FontCache, netprofm, nsi,

RemoteRegistry, W32Time, WinHttpAutoProxySv svchost.exe
392 CryptSvc, Dnscache, LanmanWorkstation, NlaSvc, WinRM
svchost.exe               1064 BFE, DPS, MpsSvc

spoolsv.exe               1388 Spooler Microsoft.ActiveDirectory 1412
ADWS

svchost.exe               1460 AppHostSvc dfsrs.exe              1492
DFSR
dns.exe                   1532 DNS
inetinfo.exe              1572 IISADMIN

ismserv.exe               1624 IsmServ svchost.exe              1752
W3SVC, WAS
```

By default, the output is formatted as a table, and you cannot use the list or *CSV* format. Beyond formatting, the important thing to note here is that services are listed by their abbreviated names, which is the naming style used by Sc, the service controller command-line utility, to manage services.

You can use the correlation between processes and services to help you manage systems. For example, if you think you are having problems with the World Wide Web Publishing Service (W3svc), one step in your troubleshooting process is to begin monitoring the service's related process or processes. You would want to examine the following:

- Process status
- Memory usage
- CPU time

By tracking these statistics over time, you can watch for changes that could indicate the process has stopped responding or is a runaway process hogging CPU time, or that there is a memory leak.

Viewing Lists of DLLs Being Used by Processes

When you use Tasklist with the /M parameter, you can examine the relationship between running processes and DLLs configured on the system. In the output, you'll see the process image name, process ID, and a list of all DLLs that the process is using, as shown in the following example:

```
Image Name                           PID Modules
===========                    === =======


System Idle Process              0 N/A System

4 N/A smss.exe                       288 N/A

csrss.exe                      404 N/A csrss.exe

456 N/A wininit.exe                    464 N/A

winlogon.exe                   508 N/A taskhostex.exe
3180 ntdll.dll, KERNEL32.DLL, KERNELBASE.dll, msvcrt.dll, RPCRT4.dll,
combase.dll, OLEAUT32.dll, kernel.appcore.dll, CRYPTBASE.dll
rdpclip.exe                   3248 ntdll.dll, KERNEL32.DLL,
KERNELBASE.dll, ADVAPI32.dll, USER32.dll, msvcrt.dll, SHELL32.dll,
WINSTA.dll, WTSAPI32.dll, ole32.dll, MPR.dll
explorer.exe                  3332 ntdll.dll, KERNEL32.DLL,
KERNELBASE.dll, apphelp.dll, msvcrt.dll, OLEAUT32.dll, combase.dll,
powrprof.dll, advapi32.dll, USER32.dll, GDI32.dll
```

```
ServerManager.exe             3860 N/A cmd.exe
3940 ntdll.dll, KERNEL32.DLL, KERNELBASE.dll, msvcrt.dll, winbrand.dll
conhost.exe                   3976 ntdll.dll, KERNEL32.DLL,
KERNELBASE.dll, GDI32.dll, USER32.dll, msvcrt.dll, IMM32.dll,
OLEAUT32.dll, combase.dll, MSCTF.dll, RPCRT4.dll, uxtheme.dll,
dwmapi.dll, comctl32.DLL, ole32.dll, sechost.dll
```

Knowing which DLL modules a process has loaded can further help you pinpoint what may be causing a process to become nonresponsive, to fail to release the CPU, or to use more memory than it should. In some cases, you might want to check DLL versions to ensure that they are the correct DLLs that the system should be running. Here, you would need to consult the Microsoft Knowledge Base or manufacturer documentation to verify DLL versions and other information.

If you are looking for processes using a specified DLL, you can also specify the name of the DLL you are looking for. For example, if you suspect that the printer spooler driver Winspool.drv is causing processes to hang up, you can search for processes that use Winspool.drv instead of Winspool32.drv and check their status and resource usage.

The syntax that you use to specify the DLL to find is `tasklist /m` *DLLName* where *DLLName* is the name of the DLL to search for. Tasklist matches the DLL name without regard to the letter case, and you can enter the DLL name in any letter case. Consider the following example: `tasklist /m winspool.drv` In this example, you are looking for processes using Winspool.drv. The output of the command would show the processes using the DLL along with their process IDs, as shown in the following example:

```
Image Name                      PID Modules
===========                     === =======

rdpclip.exe                   3248 WINSPOOL.DRV
explorer.exe                  3332 WINSPOOL.DRV
```

Filtering Task List Output

Using the /Fi parameter of the Tasklist utility, you can filter task lists using any of the information fields available, even if the information field isn't normally included in the output because of the parameters you've normally included in the output because of the parameters you've specified. This means you can specify that you want to see only processes listed with a status of Not Responding, only information for Svchost.exe processes, or only processes that use a large amount of CPU time.

You designate how a filter should be applied to a particular Tasklist information field using filter operators. The following filter operators are available:

- **Eq** Equals. If the field contains the specified value, the process is included in the output.
- **Ne** Not equals. If the field contains the specified value, the process is excluded from the output.
- **Gt** Greater than. If the field contains a numeric value and that value is greater than the value specified, the process is included in the output.
- **Lt** Less than. If the field contains a numeric value and that value is less than the value specified, the process is included in the output.
- **Ge** Greater than or equal to. If the field contains a numeric value and that value is greater than or equal to the value specified, the process is included in the output.
- **Le** Less than or equal to. If the field contains a numeric value and that value is less than or equal to the value specified, the process is included in the output.

As Table 7-1 shows, the values that you can use with filter operators depend on the task list information field you use. Remember that all fields are available even if they aren't normally displayed with the parameters you've specified. For example, you can match the status field without using the /V (verbose) flag.

TABLE 7-1 Filter Operators and Valid Values for Tasklist

FILTER FIELD NAME	VALID OPERATORS	VALID VALUES

CPUTime

eq, ne, gt, lt, ge, le Any valid time in the format hh:mm:ss Services

eq, ne

Any valid string of characters ImageName

eq, ne

Any valid string of characters MemUsage

eq, ne, gt, lt, ge, le Any valid integer, expressed in kilobytes (KB) Modules

eq, ne

DLL name

PID

eq, ne, gt, lt, ge, le Any valid positive integer Session

eq, ne, gt, lt, ge, le Any valid session number SessionName

eq, ne

Any valid string of characters Status

eq, ne

Running, Not Responding, Unknown Username

eq, ne

Any valid user name, with user name only or in domain\user format WindowTitle

eq, ne

Any valid string of characters You must use double quotation marks to enclose the filter string. Consider the following examples to see how you can use filters: Look for processes that are not responding: `tasklist /fi "status eq not responding"`

When working with remote systems, you can't filter processes by status or window title. A workaround for this in some cases is to pipe the output through the FIND command, such as **tasklist /v *s Mailer1* u imaginedlands\wrstanek | find /i “not responding”**. Note that in this case, the field you are filtering must be in the output, which is why the /V parameter was added to the example. Further, you should specify that the FIND command should ignore the letter case of characters by using the /I parameter.

Look for processes on Mailer1 with a CPU time of more than 30 minutes:
```
tasklist s Mailer1 fi "cputime gt 00:30:00"
```

Look for processes on Mailer1 that use more than 20,000 KB of memory:
```
tasklist s Mailer1 u imaginedlands\wrstanek /fi "memusage gt 20000"
```

Enter multiple /Fi “Filter” parameters to specify that output must match against multiple filters:
```
tasklist s Mailer1 fi "cputime gt 00:30:00" /fi
"memusage gt 20000"
```

Monitoring System Resource Usage and Processes

When you are working with processes, you’ll often want to get a snapshot of system resource usage, which will show you exactly how memory is being used. One way to get such a snapshot is to use the Typeperf command to display current values for key counters of the memory object. As discussed in Chapter 7, “Event Logging, Tracking, and Monitoring,” the Memory object is one of many performance objects available, and you can list its related performance counters by typing **typeperf -q Memory** at a command line.

Table 7-2 provides a summary of key counters of the Memory object. Most counters of the Memory object display the last observed value or the current percentage value rather than an average.

TABLE 7-2 Key Counters of the Memory Object

MEMORY OBJECT COUNTER	COUNTER DESCRIPTION
% Committed Bytes In Use	The ratio of Committed Bytes to the Commit Limit. Committed memory is the physical memory in use for which space has been reserved in the paging file if it needs to be written to disk. The commit limit is determined by the size of the paging file. If

Windows increases the paging file size, the commit limit increases as well, and the ratio is reduced.

Available MBytes

The amount of physical memory, in megabytes, immediately available for allocation to a process or for system use. This is physical memory not currently being used and available for use. It is equal to the sum of memory assigned to the standby (cached), free, and zero page lists. When less than five percent of memory is free, the system is low on memory and performance can suffer.

Cache Bytes

The sum of the System Cache Resident Bytes, System Driver Resident Bytes, System Code Resident Bytes, and Pool Paged Resident Bytes counters. This provides information on the memory used by the operating system kernel. Critical portions of kernel memory must operate in physical memory and can't be paged to virtual memory; the rest of kernel memory can be paged to virtual memory.

Cache Bytes Peak

The maximum number of bytes used by the file system cache since the system was last restarted.

Cache

Faults/sec The rate at which faults occur when a page sought in the file system cache is not found and must be retrieved from elsewhere in memory (a soft fault) or from disk (a hard fault). The file system cache is an area of physical memory that stores recently used pages of data for applications.

Commit Limit

The amount of virtual memory, measured in bytes, that can be committed without having to extend the paging file(s). As the number of committed bytes grows, the paging file is allowed to grow up to its maximum size, which can be determined by subtracting the total physical memory on the system from the commit limit. If you set the initial paging file size too small, the system will repeatedly extend the paging file and this requires system resources. It is better to set the initial page size as appropriate for typical usage or simply use a fixed paging file size.

Committed Bytes

The amount of committed virtual memory, in bytes. Committed memory is the physical memory in use for which space has been reserved in the paging file if it needs to be written to disk. Each physical drive can have one or more paging files. If a system is using too much virtual memory relative to the total physical memory on the system, you might need to add physical memory.

Demand Zero Faults/sec The rate at which a zeroed page is required to satisfy a fault, according to the difference between the values observed in the last two samples, divided by the duration of the sample interval. Pages emptied of previously stored data and filled with zeros are a security feature of Windows that prevent processes from seeing data stored by earlier processes that used the memory space.

Free & Zero Page List Bytes The amount of physical memory, in bytes, that is assigned to the free and zero page lists. This memory does not contain cached data and is immediately available for allocation to a process or for system use.

Free System Page Table Entries The number of page table entries not currently in use by the system.

Modified Page List Bytes The amount of physical memory, in bytes, that is assigned to the modified page list. Areas of memory on the modified page list contain cached data and code that is not actively in use by processes, the system, or the system cache. Windows needs to write out this memory before it will be available for allocation to a process or for system use.

Page

Faults/sec The number of pages faulted per second. This counter includes both hard and soft faults. Soft faults result in memory lookups. Hard faults require access to disk.

Page

Reads/sec The number of read operations required per second to resolve hard page faults. Hard page faults occur when a requested page isn't in memory and the computer has to go to disk to get it. Too many hard faults can cause significant delays and hurt performance.

Page

Writes/sec The number of page writes to disk to free up space in physical memory. Pages are written to disk only if they are changed while in physical memory.

Pages

Input/sec The rate at which pages are read from disk to resolve hard page faults. Hard page faults occur when a requested page isn't in memory and the computer has to go to disk to get it. Too many hard faults can cause significant delays and hurt performance.

Pages

Output/sec The rate at which pages are written to disk to free up space in physical memory. If the computer has to free up memory too often, this is an indicator that the system doesn't have enough physical memory (RAM).

Pages/sec

The number of memory pages that are read from disk or written to disk to resolve hard page faults. It is the sum of Pages Input/sec and Pages Output/sec.

Pool Nonpaged Allocs The number of calls to allocate space in the nonpaged pool. The nonpaged pool is an area of system memory area for objects that cannot be written to disk, and must remain in physical memory as long as they are allocated.

Pool Nonpaged Bytes The size, in bytes, of the nonpaged pool, an area of system memory for objects that cannot be written to disk, but must remain in physical memory as long as they are allocated. If the size of the nonpaged pool is large relative to the total amount of virtual memory allocated to the computer, you might want to increase the virtual memory size. If this value slowly increases in size over time, a kernel mode process might have a memory leak.

Pool Paged Allocs

The number of calls to allocate space in the paged pool. The paged pool is an area of system memory for objects that can be written to disk when they are not being used.

Pool Paged Bytes

The total size, in bytes, of the paged pool. The paged pool is an area of system memory for objects that can be written to disk when they are not being used. If the size of the paged pool is large relative to the total amount of physical memory on the system, you might need to add memory to the system. If this value slowly increases in size over time, a kernel mode process might have a memory leak.

Pool Paged Resident Bytes The size, in bytes, of the paged pool that is currently resident and actively being used. Typically, the resident bytes in the paged pool is a smaller amount than the total bytes assigned to the paged pool.

Standby Cache Core Bytes The amount of physical memory, in bytes, that is assigned to the core standby cache page lists. A standby cached page list is an area of memory that contains cached data and code that is not actively in use by processes, the system, or the system cache. It is immediately available for allocation to a process or for system use. If the system runs out of available free–and-zero memory, memory on lower priority standby cache page lists will be repurposed before memory on higher priority standby cache page lists.

System Cache Resident Bytes The size, in bytes, of the pageable operating system code in the file system cache. This value includes only current physical pages and does not include any virtual memory pages not currently resident.

System Code Resident Bytes The size, in bytes of the operating system code currently in physical memory that can be written to disk when not in use.

System Code Total Bytes The size, in bytes, of the pageable operating system code currently in virtual memory. It is a measure of the amount of physical memory being used by the operating system that can be written to disk when not in use and does not include code that must remain in physical memory and cannot be written to disk.

System Driver Resident Bytes The size, in bytes, of the pageable physical memory being used by device drivers. It is the working set (physical memory area) of the drivers.

System Driver Total Bytes The size, in bytes, of the pageable memory and pageable virtual memory currently being used by device drivers. It includes physical memory and code and data paged to disk.

Transition Faults/sec The rate at which page faults are resolved by recovering pages that were being used by another process sharing the page, or were on the modified page list or the standby list, or were being written to disk at the time of the page fault. The pages were recovered without additional disk activity.

Transition Pages RePurposed/sec The rate at which the number of transition cache pages were reused for a different purpose. These pages would have otherwise remained in the page cache to provide a soft fault in the event the page was accessed in the future. These pages can contain private or sharable memory.

Write

Copies/sec The rate at which page faults are caused by attempts to write that have been satisfied by copying the page from elsewhere in physical memory.

Sample 7-1 provides an example of how you can use Typeperf to get a snapshot of memory usage. In this example, you use a counter file called Perf.txt to specify the counters you want to track. You collect five samples with an interval of 30 seconds between samples and save the output in a file called SaveData.txt. If you import the data into a spreadsheet or convert it to a table in a Microsoft Office Word document, you can make better sense of the output and will know exactly how the computer is using memory.

> *NOTE* I chose to track these counters because they give you a good overall snapshot of memory usage. If you save the command line as a script, you can run the script as a scheduled task to get a snapshot of memory usage at various times of the day.

SAMPLE 7-1 Getting a snapshot of memory usage Command-line

```
typeperf -cf c:\logs\perf.txt –o c:\logs\savedata.txt -sc 5 -si 30
```

Source for Perf.txt

```
\memory\% Committed Bytes In Use
\memory\Available MBytes
\memory\Cache Bytes
\memory\Cache Bytes Peak
\memory\Committed Bytes
\memory\Commit Limit
\memory\Page Faults/sec
\memory\Pool Nonpaged Bytes
```

```
\memory\Pool Paged Bytes
```
Sample output

```
"(PDH-CSV 4.0)","\\SERVER12\memory\% Committed Bytes In Use","
\\SERVER12\memory\Available MBytes","\\SERVER12\memory\Cache
Bytes","\\SERVER12\memory\Cache Bytes Peak","\\SERVER12\memory\Committed
Bytes","
\\SERVER12\memory\Commit Limit","\\SERVER12\memory\Page
Faults/sec","\\SERVER12\memory\Pool Nonpaged
Bytes","\\SERVER12\memory\Pool Paged Bytes"
"03/25/2015 14:24:28.033","22.860837","2023.000000","260632576.000000","
280514560.000000","1636175872.000000","7157112832.000000","80.494007","7
"03/25/2015 14:24:30.033","22.861294","2023.000000","260653056.000000","
280514560.000000","1636208640.000000","7157112832.000000","70.997253","7
"03/25/2015 14:24:32.033","22.861294","2023.000000","260653056.000000","
280514560.000000","1636208640.000000","7157112832.000000","3.000142","73
"03/25/2015 14:24:34.033","22.861581","2023.000000","260673536.000000","
280514560.000000","1636229120.000000","7157112832.000000","15.999741","7
"03/25/2015 14:24:36.033","22.861695","2023.000000","260681728.000000","
280514560.000000","1636237312.000000","7157112832.000000","6.499981","73
```

You can obtain detailed information about running processes using the Windows PowerShell Get-Process cmdlet. See Table 7-3 for a summary of this cmdlet's significant properties. At a Windows PowerShell prompt, you can view important statistics for all processes by following these steps: 1. Get all the processes running on the server and store them in the $a variable by entering: `$a =  get-process`

2. Use the InputObject parameter to pass the process objects stored in $a to Get-Process and then pass the objects to the format-table cmdlet along with the list of properties you want to see by entering:

```
get-process -inputobject $a | format-table –property ProcessName,
BasePriority, HandleCount, Id,
NonpagedSystemMemorySize, PagedSystemMemorySize,
PeakPagedMemorySize, PeakVirtualMemorySize,
PeakWorkingSet, SessionId, Threads, TotalProcessorTime,
VirtualMemorySize, WorkingSet, CPU, Path
```

The order of the properties in the comma-separated list determines the display order. If you want to change the display order, simply move the property to a different position in the list.

When you know the process you want to examine, you don't need to use this multistep procedure. Simply enter the name of the process without the .exe or .dll instead of using —inputobject $a. In the following example, you list details about the winlogon process: `get-process winlogon | format-table –property ProcessName, BasePriority, HandleCount, Id, NonpagedSystemMemorySize, PagedSystemMemorySize, PeakPagedMemorySize, PeakVirtualMemorySize, PeakWorkingSet, SessionId, Threads, TotalProcessorTime, VirtualMemorySize, WorkingSet, CPU, Path` You can enter part of a process name as well using an asterisk as a wildcard to match a partial name. In this example, Get-Process lists any process with a name that starts with winl: `get-process winl* | format-table –property ProcessName, BasePriority, HandleCount, Id, NonpagedSystemMemorySize, PagedSystemMemorySize, PeakPagedMemorySize, PeakVirtualMemorySize, PeakWorkingSet, SessionId, Threads, TotalProcessorTime, VirtualMemorySize, WorkingSet, CPU, Path` *TIP* By default, many properties that measure memory usage are defined as 32-bit values. When working with Get-Process on 64-bit systems, you'll find that these properties have both a 32-bit and 64-bit version. On 64-bit systems with more than 4 GB of RAM, you'll need to use the 64-bit versions to ensure you get accurate values.

TABLE 7-3 Properties of Get-Process and How They Are Used

PROPERTY NAME	PROPERTY DESCRIPTION
BasePriority	Shows the priority of the process. Priority determines how much of the system resources are allocated to a process. The standard priorities are Low (4), Below Normal (6), Normal (8), Above Normal (10), High (13), and Real-Time (24). Most processes have a Normal priority by default, and the highest priority is given to real-time processes.
CPU	Shows the percentage of CPU utilization for the process. The System Idle Process shows what percentage of CPU power is idle. A value of 99 for the System Idle Process means 99 percent of the system resources currently aren't being used. If the system has low idle time (meaning high CPU usage) during peak or average usage, you might consider upgrading to faster processors or adding processors.
Description	Shows a description of the process.
FileVersion	Shows the file version of the process's executable.
HandleCount	Shows the number of file handles maintained by the process. The number of handles used is an indicator of how dependent the process is on the file system. Some processes have thousands of open file handles. Each file handle requires system memory to maintain.
Id	Shows the run-time identification number of the process.
MinWorkingSet	Shows the minimum amount of working set memory used by the process.

Modules

Shows the executables and dynamically linked libraries used by the process.

NonpagedSystemMemorySize / NonpagedSystemMemorySize64

Shows the amount of virtual memory for a process that cannot be written to disk. The nonpaged pool is an area of RAM for objects that can't be written to disk. You should note processes that require a high amount of nonpaged pool memory. If the server doesn't have enough free memory, these processes might be the reason for a high level of page faults.

PagedSystemMemorySize / PagedSystemMemorySize64

Shows the amount of committed virtual memory for a process that can be written to disk. The paged pool is an area of RAM for objects that can be written to disk when they aren't used. As process activity increases, so does the amount of pool memory the process uses. Most processes have more paged pool than nonpaged pool requirements.

Path

Shows the full path to the executable for the process.

PeakPageMemorySize / PeakPageMemorySize64

Shows the peak amount of paged memory used by the process.

PeakVirtualMemorySize / PeakVirtualMemorySize64

Shows the peak amount of virtual memory used by the process.

PeakWorkingSet / PeakWorkingSet64

Shows the maximum amount of memory the process used, including both the private working set and the non-private working set. If peak memory is exceptionally large, this can be an indicator of a memory leak.

PriorityBoostEnabled

Shows a Boolean value that indicates whether the process has the PriorityBoost feature enabled.

PriorityClass

Shows the priority class of the process.

PrivilegedProcessorTime

Shows the amount of kernel-mode usage time for the process.

ProcessName

Shows the name of the process.

ProcessorAffinity

Shows the processor affinity setting for the process.

Responding

Shows a Boolean value that indicates whether the process responded when tested.

SessionId

Shows the identification number user (session) within which the process is running. This corresponds to the ID value listed on the Users tab in Task Manager.

StartTime

Shows the date and time the process was started.

Threads

Shows the number of threads that the process is using. Most server applications are multithreaded, which allows concurrent execution of process requests. Some applications can dynamically control the number of concurrently executing threads to improve application performance. Too many threads, however, can actually reduce performance, because the operating system has to switch thread contexts too frequently.

TotalProcessorTime

Shows the total amount of CPU time used by the process since it was started. If a process is using a lot of CPU time, the related application might have a configuration problem. This could also indicate a runaway or nonresponsive process that is unnecessarily tying up the CPU.

UserProcessorTime

Shows the amount of user-mode usage time for the process.

VirtualMemorySize / VirtualMemorySize64

Shows the amount of virtual memory allocated to and reserved for a process. Virtual memory is memory on disk and is slower to access than pooled memory. By configuring an application to use more physical RAM, you might be able to increase performance. To do this, however, the system must have available RAM. If it doesn't, other processes running on the system might slow down.

WorkingSet / WorkingSet64

Shows the amount of memory the process is currently using, including both the private working set and the non-private working set. The private working set is memory the process is using that cannot be shared with other processes. The non-private working set is memory the process is using that can be shared with other processes. If memory usage for a process slowly grows over time and doesn't go back to the baseline value, this can be an indicator of a memory leak.

Stopping Processes

When you want to stop processes that are running on a local or remote system, you can use the Taskkill command-line utility. With Taskkill, you can stop processes by process ID using the /Pid parameter or image name using the *Im parameter. If you want to stop multiple processes by process ID or image name, you can enter multiple* Pid or /Im parameters as well. With image names, however, watch out, because Taskkill will stop all processes that have that image name. Thus if three instances of Helpctr.exe are running, all three processes would be stopped if you use Taskkill with that image name.

As with Tasklist, Taskkill runs by default with the permissions of the user who is currently logged on, and you can also specify the remote computer whose tasks you want to query, and the Run As permissions. To do this, you use the expanded syntax, which includes the following parameters: `/s Computer u [Domain\]User [p Password]`

where *Computer* is the remote computer name or IP address, *Domain* is the optional domain name in which the user account is located, *User* is the name of the user account whose permissions you want to use, and *Password* is the optional password for the user account. If you don't specify the domain, the current domain is assumed. If you don't provide the account password, you are prompted for the password.

> *NOTE* Sometimes it is necessary to force a process to stop running. Typically, this is necessary when a process stops responding while opening a file, reading or writing data, or performing other read/write operations. To force a process to stop, you use the /F parameter. This parameter is only used with processes running on local systems. Processes stopped on remote systems are always forcefully stopped.

> *TIP* As you examine processes, keep in mind that a single application might start multiple processes. Generally, these processes depend on a central process, and from this main process a process tree containing dependent processes is formed. Occasionally, you may want to stop the entire process tree, starting with the parent application process and including any dependent processes. To do this, you can use the /T parameter.

Consider the following examples to see how you can use Taskkill: Stop process ID 208:

```
taskkill /pid 208
```

Stop all processes with the image name Cmd.exe: `taskkill /im cmd.exe`
Stop processes 208, 1346, and 2048 on MAILER1: `taskkill s Mailer1 pid 208 pid 1346 pid 2048`

Force local process 1346 to stop: `taskkill f pid 1346`

Stop a process tree, starting with process ID 1248 and including all child processes: `taskkill t pid 1248`

To ensure that only processes matching specific criteria are stopped, you can use all the filters listed in Table 7-1 except SessionName. For example, you can use a filter to specify that only instances of Cmd.exe that are not responding should be stopped rather than all instances of Cmd.exe (which is the default when you use the /Im parameter).

As with Tasklist, Taskkill provides a Modules filter with operators EQ and

NE to allow you to specify DLL modules that should be excluded or included. As you may recall, you use the Tasklist /M parameter to examine the relationship between running processes and DLLs configured on the system. Using the Taskkill Modules filter with the EQ operator, you could stop all processes using a specific DLL. Using the Taskkill Modules filter with the NE operator, you ensure that processes using a specific DLL are not stopped.

> *TIP* When you use filters, you don't have to specify a specific image name or process ID to work with. This means you can stop processes based solely on whether they match filter criteria. For example, you can specify that you want to stop all processes that aren't responding.

As with Tasklist, you can also use multiple filters. Again, you must use double quotation marks to enclose the filter string. Consider the following examples to see how you can use filters with Taskkill: Stop instances of Cmd.exe that are not responding: `taskkill im cmd.exe fi "status eq not responding"`

Stop all processes with a process ID greater than 4 if they aren't responding: `taskkill fi "pid gt 4" fi "status eq not responding"`

Stop all processes using the Winspool.drv DLL: `taskkill /fi "modules eq winspool.drv"`

Although the /Im and /Pid flags are not used in the second example, the process IDs are filtered so that only certain processes are affected. You don't want to stop the system or system idle process accidentally. Typically, these processes run with process IDs of 4 and 0 respectively, and if you stop them, the system will stop responding or shut down.

Detecting and Resolving Performance Issues Through Monitoring

At the command line, Tasklist and Windows PowerShell Get-Process provide everything you need for detecting and resolving most performance issues. However, you'll often need to dig deep to determine whether a problem exists and if so, what is causing the problem.

Monitoring Memory Paging and Paging to Disk

Often, you'll want to get detailed information on hard and soft page faults that are occuring. A page fault occurs when a process requests a page in memory and the system can't find it at the requested location. If the requested page is elsewhere in memory, the fault is called a soft page fault. If the requested page must be retrieved from disk, the fault is called a hard page fault.

To see page faults that are occurring in real time, enter the following at the command line: `typeperf "\memory\Page Faults/sec" -si 5`

To stop Typeperf, press Ctrl+C. Page faults are shown according to the number of hard and soft faults occuring per second. Other counters of the Memory object that you can use for tracking page faults include:

- Cache Faults/sec
- Demand Zero Faults/sec
- Page Reads/sec
- Page Writes/sec
- Write Copies/sec
- Transition Faults/sec
- Transition Pages RePurposed/sec

Pay particular attention to the Page Reads/sec and Page Writes/sec, which provide information on hard faults. Although developers will be interested in the source of page faults, administrators are more interested in how many page faults are occurring. Most processors can handle large numbers of soft faults. A soft fault simply means the system had to look elsewhere in memory for the requested memory page. With a hard fault, on the other hand, the requested memory page must be retrieved from disk, which can cause significant delays. If you are seeing a lot of hard faults, you may need to increase the amount of memory or reduce the amount of memory being cached by the system and applications.

In addition to counters of the Memory object discussed previously, you can use the following objects and counters to check for disk paging issues:

- **Paging File(*)\% Usage** The percentage of the paging file currently in use. If this value approaches 100 percent for all instances, you should consider either increasing the virtual memory size or adding physical memory to the system. This will ensure that the computer has additional memory if it needs it, such as when the computer load grows.
- **Paging File(*)\% Usage Peak** The peak size of the paging file as a percentage of the total paging file size available. A high value can mean that the paging file isn't large enough to handle increased load conditions.
- **PhysicalDisk(*)\% Disk Time** The percentage of time that the selected disk spent servicing read and write requests. Keep track of this value for the physical disks that have paging files. If you see this value increasing over several monitoring periods, you should more closely monitor paging file usage and you might consider adding physical memory to the system.
- **PhysicalDisk(*)\Avg. Disk Queue Length** The average number of read and write requests that were waiting for the selected disk during the sample interval. Keep track of this value for the physical disks that have paging files. If you see this value increasing over time and the Memory\Page Reads/Sec is also increasing, the system is having to perform a lot of paging file reads.

The asterisks in parentheses are placeholders for the object instance. If a particular object has multiple instances, such as when a computer has multiple physical disks or multiple paging files, you can use an object instance to track a specific occurrence of that object. You could also elect to track all instances of an object, such as whether you want to monitor all physical disks on a system. Specify _Total to work with all counter instances, or specify individual counter instances to monitor.

Sample 7-2 provides an example of how you can use Typeperf to get a snapshot of disk paging. In this example, you use a counter file called PagePerf.txt to specify the counters you want to track. You collect five samples with an interval of 30 seconds between samples and save the output in a file called SavePageData.txt. If you import the data into a spreadsheet or convert it to a table in a Word document, you can make better sense of the output and a better understanding of how the computer is using the page file and paging to disk.

SAMPLE 7-2 Checking disk paging Command line

```
typeperf -cf c:\logs\pageperf.txt -o c:\logs\savepagedata.txt -sc 5
-si 30
```

Source for PagePerf.txt

```
\memory\Pages/Sec
\Paging File(_Total)\% Usage
\Paging File(_Total)\% Usage Peak
\PhysicalDisk(_Total)\% Disk Time
\PhysicalDisk(_Total)\Avg. Disk Queue Length
```

Monitoring Memory Usage and the Working Memory Set for Individual Processes

You can use Tasklist to get basic memory usage for a process. The syntax you can use is: `tasklist /fi "pid eq ProcessID"`

where *ProcessID* is the id number of the process you want to work with. The output from Tasklist will show you how much memory the process is currently using. For example, if you were tracking process ID 7292, your output might look like the following:

```
Image Name      PID      Session
Name     Session#    Mem Usage
============= ====     ============     ========    ==========
jvappm.exe     7292                             1      7,424 K
```

In this example, the process is using 7,424 KB of memory. By watching the memory usage over time, you can determine whether the memory usage is increasing. If memory usage is increasing compared to a typical baseline, the process might have a memory-related problem.

Sample 7-3 provides the source for a command-line script that checks the memory usage of a process over a timed interval. The script expects the process ID you want to work with to be passed as the first parameter. If you do not supply a process ID, error text is written to the output.

SAMPLE 7-3 Viewing memory usage at the command line
MemUsage.bat

```
@echo off
if "%1"=="" (echo Error: please enter Process ID to track) & (goto EXIT)

tasklist /fi "pid eq %1"
timeout /t 600
tasklist /fi "pid eq %1"
timeout /t 600
tasklist /fi "pid eq %1"
:EXIT
```

Sample output

```
Image Name           PID Session Name      Session#   Mem Usage
==========          ==== ============      ========   =========
jvapm.exe           7292                        1      7,452 K

Waiting for  0 seconds, press a key to continue ...

Image Name           PID Session Name      Session#   Mem Usage
==========          ==== ============      ========   =========
jvapm.exe            7292                         1     7,452 K

Waiting for  0 seconds, press a key to continue ...

Image Name           PID Session Name      Session#   Mem Usage
==========          ==== ============      ========   =========
jvapm.exe            7292                         1     7,452 K
```

In Sample 7-3, the process's memory usage does not change over the sampled interval. Because of this, it is unlikely the process has a memory leak, but to be sure you'd need to sample over a longer period.

You can use the Windows PowerShell Get-Process cmdlet to track detailed memory usage for individual processes. The syntax you can use is `get-process ProcessName | format-table –property NonpagedSystemMemorySize, PagedSystemMemorySize, VirtualMemorySize, PeakVirtualMemorySize, MinWorkingSet, WorkingSet, PeakWorkingSet` where ProcessName is the name of the process without the .exe or .dll. In a Windows PowerShell script, such as the one shown as Sample 7-4, you could combine the Get-Process cmdlet with the start-sleep cmdlet to view the memory usage for a process at timed intervals.

SAMPLE 7-4 Viewing detailed memory usage MemUsage.ps1

```
get-process msdtc | format-table –property NonpagedSystemMemorySize,
PagedSystemMemorySize, VirtualMemorySize, PeakVirtualMemorySize,
MinWorkingSet, WorkingSet, PeakWorkingSet

start-sleep –seconds 600

get-process msdtc | format-table –property NonpagedSystemMemorySize,
PagedSystemMemorySize, VirtualMemorySize, PeakVirtualMemorySize,
MinWorkingSet, WorkingSet, PeakWorkingSet

start-sleep –seconds 600

get-process msdtc | format-table –property NonpagedSystemMemorySize,
PagedSystemMemorySize, VirtualMemorySize, PeakVirtualMemorySize,
MinWorkingSet, WorkingSet, PeakWorkingSet
```

Sample output

```
Nonpaged PagedSystem     Virtual       Peak    Working       Peak
 System      Memory      Memory    Virtual        Set    Working
 MemorySize    Size        Size MemorySize                   Set
----------- --------- ----------- -------- ---------- ----------
       6304     70544   41766912  63631360    6287360    6344704
   Nonpaged PagedSystem     Virtual       Peak    Working       Peak

Nonpaged PagedSystem     Virtual       Peak    Working       Peak
 System      Memory      Memory    Virtual        Set    Working
 MemorySize    Size        Size MemorySize                   Set
----------- --------- ----------- -------- ---------- ----------
       8123     96343   56243535  97423424    9147256    9348942

Nonpaged PagedSystem     Virtual       Peak    Working       Peak
 System      Memory      Memory    Virtual        Set    Working
 MemorySize    Size        Size MemorySize                   Set
----------- --------- ----------- -------- ---------- ----------
      17564    129645   48934246  97423424    9987384   10344706
```

NOTE Windows PowerShell script files have the .ps1 filename extension. To run a script at the Windows PowerShell prompt, you type the name of the script and, optionally, the filename extension. You must specify the fully qualified path to the script file, even if the script is in the current directory. To indicate the current directory, type the directory name or use the dot (.) to represent the current directory. With the MemUsage.ps1 script in the current directory, you can run the script by entering **.\memusage.ps1** at the Windows PowerShell prompt.

The Get-Process properties examined in Sample 7-4 provide the following information:

- **NonPagedSystemMemorySize** Shows the amount of allocated memory that can't be written to disk
- **PagedSystemMemorySize** Shows the amount of allocated memory that is allowed to be paged to the hard disk
- **VirtualMemorySize** Shows the amount of virtual memory allocated to and reserved for a process
- **PeakVirtualMemorySize** Shows the peak amount of paged memory used by the process
- **WorkingSetSize** Shows the amount of memory allocated to the process by the operating system
- **PeakWorkingSet** Shows the peak amount of memory used by the process

When you focus on these properties, you are zeroing in on the memory usage of a specific process. The key aspect to monitor is the working memory set. The working set of memory shows how much memory is allocated to the process by the operating system. If the working set increases over time and doesn't eventually go back to baseline usage, the process may have a memory leak. With a memory leak, the process isn't properly releasing memory that it's using, which can lead to reduced performance of the entire system.

In Sample 7-4, the process's memory usage changes substantially over the sampled interval. While it is most likely the process is simply actively being used by users or the computer itself, the process should eventually being used by users of the computer itself, the process should eventually return to a baseline memory usage. If this doesn’t happen, the process may have a memory-related problem.

Resolving Performance Bottlenecks

Because memory is usually the primary performance bottleneck on both workstations and servers, I’ve discussed many techniques previously in this chapter that you can use to help identify problems with memory. Memory is the resource you should examine first to try to determine why a system isn’t performing as expected.

However, memory isn’t the only bottleneck. Processor bottlenecks can occur if a process’s threads need more processing time than is available. If a system’s processors are the performance bottleneck, adding memory, drives, or network connections won’t solve the problem. Instead, you might need to upgrade the processors to faster clock speeds or add processors to increase the computer’s upper capacity. With servers, you could also move processor-intensive applications to another server.

Typeperf counters you can use to check for processor bottlenecks include the following:

- **System\Processor Queue Length** Records the number of threads waiting to be executed. These threads are queued in an area shared by all processors on the system. The processor queue grows because threads have to wait to get processing time. As a result, the system response suffers and the system appears sluggish or nonresponsive. Here, you might need to upgrade the processors to faster clock speeds or add processors to increase the server’s upper capacity.
- **Processor(*)\% Processor Time** Records the percentage of time the selected processor is executing a non-idle thread. You should track this counter separately for each processor instance on the server. If the % Processor Time values for all instances are

high (above 75 percent) while the network interface and disk input/output (I/O) throughput rates are relatively low, you might need to upgrade the processors to faster clock speeds or add processors to increase the server's upper capacity.

- **Processor(*)\% User Time** Records the percentage of time the selected processor is executing a non-idle thread in *User mode*. User mode is a processing mode for applications and user-level subsystems. A high value for all processor instances might indicate that you need to upgrade the processors to faster clock speeds or add processors to increase the server's upper capacity.
- **Processor(*)\% Privileged Time** Records the percentage of time the selected processor is executing a non-idle thread in *Privileged mode*. Privileged mode is a processing mode for operating system components and services, allowing direct access to hardware and memory. A high value for all processor instances might indicate that you need to upgrade the processors to faster clock speeds or add processors to increase the computer's upper capacity.
- **Processor(*)\Interrupts/sec** Records the average rate, in incidents per second, that the selected processor received and serviced hardware interrupts. If this value increases substantially over time without a corresponding increase in activity, the system might have a hardware problem. To resolve this problem, you must identify the device or component that is causing the problem. Each time drivers or disk subsystem components such as hard disk drives or network components generate an interrupt, the processor has to stop what it is doing to handle the request because requests from hardware take priority. However, poorly designed drivers and components can generate false interrupts, which tie up the processor for no reason. System boards or components that are failing can

generate false interrupts as well.

NOTE The asterisks in parentheses are placeholders for the object instance. On multiprocessor systems, you might need to rule out processor affinity as a cause of a processor bottleneck. By using processor affinity, you can set a program or process to use a specific processor to improve its performance. Assigning processor affinity can, however, block access to the processor for other programs and processes.

A system's hard disks are rarely the primary reason for a bottleneck. If a system is having to do a lot of disk reads and writes, it is usually because there isn't enough physical memory available and the system has to page to disk. Because reading from and writing to disk is much slower than reading and writing memory, excessive paging can degrade the server's overall performance. To reduce the amount of disk activity, you want the system to manage memory as efficiently as possible and page to disk only when necessary.

You can use these counters to monitor disk reads and writes:

- **PhysicalDisk(*)\% Disk Time** Records the percentage of time the physical disk is busy. Track this value for all hard disk drives on the system in conjunction with Processor(*)\% Processor Time and Network Interface(*)\Bytes Total/sec. If the % Disk Time value is high and the processor and network connection values aren't high, the system's hard disk drives might be creating a bottleneck.
- **PhysicalDisk(*)\Current Disk Queue Length** Records the number of system requests that are waiting for disk access. A high value indicates that the disk-waits are impacting system performance. In general, you want very few waiting requests.
- **PhysicalDisk(*)\Avg. Disk Write Queue Length** Records the number of write requests that are waiting to be processed.
- **PhysicalDisk(*)\Avg. Disk Read Queue Length** Records the

number of read requests that are waiting to be processed.

- **PhysicalDisk(*)\Disk Writes/sec** Records the number of disk writes per second, which indicates the amount of disk I/O activity. By tracking the number of writes per second and the size of the write queue, you can determine how write operations are impacting disk performance.
- **PhysicalDisk(*)\Disk Reads/sec** Records the number of disk reads per second, which indicates the amount of disk I/O activity. By tracking the number of reads per second and the size of the read queue, you can determine how read operations are impacting disk performance.

Networking components can also cause bottlenecks. A delay between when a request is made, the time the request is received, and the time a user gets a response can cause users to think that systems are slow or nonresponsive. Unfortunately, in many cases, the delay users experience when working over the network is beyond your control. This is because the delay is a function of the type of connection the user has and the route the request takes. The total capacity of a computer to handle requests and the amount of bandwidth available to a computer are factors you can control, however. Network capacity is a function of the network cards and interfaces configured on the computers. Network bandwidth availability is a function of the network infrastructure and how much traffic is on it when a request is made.

You can use the following counters to check network activity and look for bottlenecks:

- **Network Interface(*)\Bytes Received/Sec** Records the rate at which bytes are received over a network adapter.
- **Network Interface(*)\Bytes Sent/Sec** Records the rate at which bytes are sent over a network adapter.
- **Network Interface(*)\Bytes Total/Sec** Records the rate at which bytes are sent and received over a network adapter.

Check the network card configuration if you think there’s a problem.

- **Network Interface(*)\Current Bandwidth** Estimates the current bandwidth for the selected network adapter in bits per second. Check to ensure the current bandwidth matches the type of network card configured on the computer. Most computers use 100 megabit, 1 gigabit, or 10 gigabit network cards. Keep in mind that if the computer has a 1 gigabit network card, the networking devices to which the computer connects must also support this speed.

Chapter 9. Scheduling Tasks to Run Automatically

As you become increasingly proficient as a user or administrator, you probably find yourself repeatedly performing the same or similar tasks every day. You may also find that you have to come in to work early or stay late to perform tasks during nonbusiness hours. These tasks might be routine maintenance activities, such as deleting temporary files so that disks don't run out of space, or backing up important data. These tasks might also be more involved processes, such as searching the event logs on all business servers for problems that need to be resolved. The good news is that if you can break down these tasks into a series of steps, chances are that you can also automate these tasks, and Windows provides a couple of ways to do this:

- **Schtasks** An advanced command-line utility for running commands, scripts, and programs on a scheduled basis. You can schedule tasks to run one time only, on a minute-by-minute basis, at a specific interval (such as hourly, daily, weekly, or monthly), at system startup, at logon, or whenever the system is idle.
- **Task Scheduler** A graphical utility for running commands, scripts, and programs on a scheduled basis. Task Scheduler performs the same operations as the Schtasks command-line utility, allowing you to use Task Scheduler and Schtasks together, and to manage tasks created in either utility by using one tool or the other.

Because you can use Schtasks and Task Scheduler interchangeably, this chapter discusses how to use both utilities to automate the running of programs, command-line utilities, and scripts. In most cases, you'll find that it is useful to understand both utilities, and that even when you use Task Scheduler for point-and-click convenience, you'll still work with the command line.

Scheduling Tasks on Local and Remote Systems

Whatever you can execute at the command line can be configured as a scheduled task, including command-line utilities, scripts, applications, shortcuts, and documents. You can also specify command-line arguments. Sometimes when you schedule tasks, you'll do so for the computer to which you are currently logged on (that is, a local system). More typically, however, when you schedule tasks you'll do so for remote systems throughout your network from your local computer (that is, a remote computer).

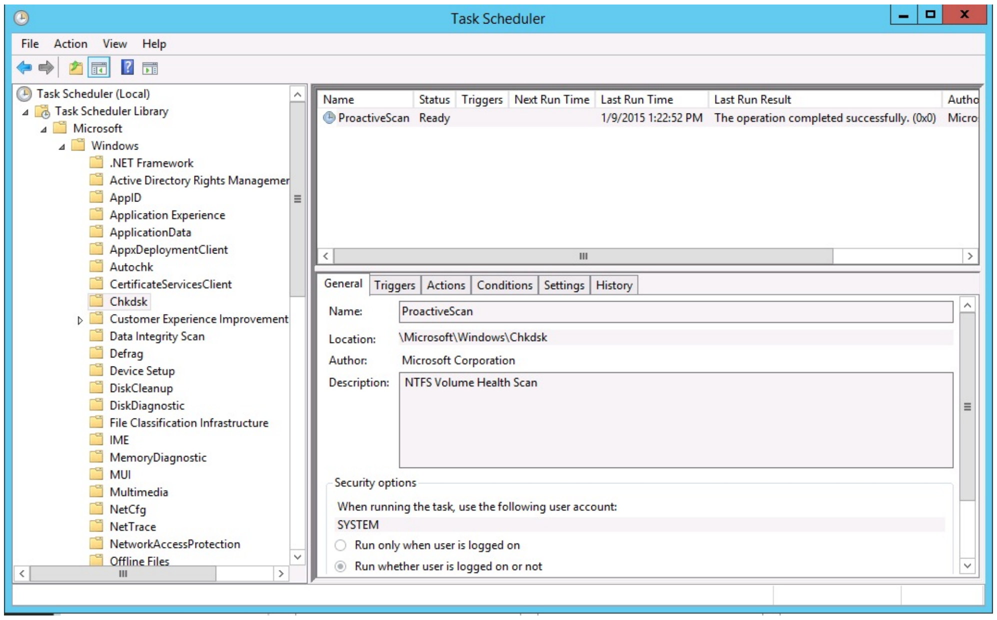
Introducing Task Scheduling

The Task Scheduler service enables local and remote task scheduling. This service must be running for each system on which you want to schedule tasks. Task Scheduler logs on as the LocalSystem account by default and usually doesn't have adequate permissions to perform administrative tasks. Because of this, you should configure each task individually to use an account with adequate user privileges and access rights to run the tasks you want to schedule. You should also make sure that the Task Scheduler service is configured to start automatically on all the systems for which you want to schedule tasks. Be sure to set the Task Scheduler startup and logon account options appropriately.

You use the Task Scheduler console to view and work with scheduled tasks. To access Task Scheduler, open Server Manager, click Tools, and then click Task Scheduler. Any user can schedule a task on the local computer, and can view and change scheduled tasks. Administrators can schedule, view, and change all tasks on the local computer except for restricted system tasks. To schedule, view, or change tasks on remote computers, you must be a member of the Administrators group on the remote computer, or you must be able to provide the credentials of an Administrator of the remote computer when prompted.

In Task Scheduler, the Task Scheduler Library contains all the tasks defined on the computer. Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 make extensive use of scheduled tasks. Many tasks are created automatically when you install and configure the operating system for the first time. When you install roles, role services, features, and applications, those components can create additional tasks as well. In Figure 9-1, the Task Scheduler Library shows the following:

- **Name** The name of the task. Task names can be any string of characters and, like other task properties, are set when you create the task.
- **Status** The current status of the task. "Running" indicates the task has been started by the task scheduler and is running. "Ready" indicates the task is enabled and ready to be triggered. "Disabled" means that task has been disabled and will not run. "Failed" indicates the task could not be started and that there is a problem with the task.
- **Triggers** Lists the triggers associated with the task.
- **Next Run Time** The next date and time the task will run. "Never" indicates the task will not run again after the scheduled run time and is probably a one-time task.
- **Last Run Time** The last date and time the task ran. "Never" indicates the task has not run for the first time.
- **Last Run Result** The exit error code. An error code of zero indicates no error occurred. Any other value indicates some type of error occurred.
- **Author** The user name of the person who created the scheduled task.
- **Created** The date and time the task was created.



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Autho
ProactiveScan	Ready			1/9/2015 1:22:52 PM	The operation completed successfully. (0x0)	Micro

Name: ProactiveScan
Location: \Microsoft\Windows\Chkdsk
Author: Microsoft Corporation
Description: NTFS Volume Health Scan

Security options
When running the task, use the following user account:
SYSTEM
Run only when user is logged on
[✓ Run whether user is logged on or not]

FIGURE 9-1 Use Task Scheduler to work with scheduled tasks in the graphical interface.

Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 have two general types of scheduled tasks:

- **Standard tasks** Automate routine tasks and perform housekeeping. These tasks are visible to users and can be modified if necessary.
- **Hidden tasks** Automate special system tasks. These tasks are hidden from users by default and should not be modified in most cases. Some hidden tasks are created and managed through a related program, such as Windows Defender.

NOTE You can display hidden tasks by selecting Show Hidden Tasks on the View menu.

In the Task Scheduler Library, you'll find system tasks under Microsoft\Windows and Microsoft\Windows Defender. Tasks under Microsoft\Windows handle many of the background housekeeping tasks on your computer. Tasks under Microsoft\Windows Defender are used to automate malware scans. Windows Defender is the default malware tool used with Windows 8.1 and can be added to Windows Server 2012 and

Windows Server 2012 R2 by installing the Desktop Experience feature.

Although you can create tasks at any level within the Task Scheduler Library, you will in most cases want to create tasks at the top level within the library. You do this by selecting the Task Scheduler node rather than a lower-level node when creating a task.

To help organize your tasks, you can create folders within the Task Scheduler Library. These folders then appear as nodes within the library hierarchy and act as containers for your tasks. In a large enterprise, you may even want to create your own subhierarchy within the Task Scheduler Library. You can create a new folder to contain your tasks by following these steps:

1. In Task Scheduler, select the top-level node that will contain your folder. For example, if you want your folder to appear as a subnode of the Task Scheduler Library node, you'd select this node.
2. On the Action menu or in the Actions pane, select New Folder. In the dialog box provided, type a unique name for the folder and then click OK.

Task Scheduler is fully integrated with the Windows security model implemented in Windows 8.1, Windows Server 2012, and Windows Server 2012 R2. When you define a task, you specify the user account under which the task runs. By default, the task runs with the user's standard privileges. If the task requires higher privileges, however, you don't want Windows to block the task and display a User Account Control prompt. For this reason, you can specify that you want Windows to always run the task with the highest user privileges available. When you run with highest user privileges, Task Scheduler prompts you for the password of the user account under which the task will run. This password is then stored securely on the system.

NOTE In the revised Task Scheduler, Microsoft has changed the way user credentials are used with tasks as well. Previously, when the password for a user account changed, you needed to modify the password settings associated with each task that used the user account. In Windows 8.1, Windows Server 2012, and Windows Server 2012 R2, you never have to update the passwords of user accounts associated with tasks that only access local resources. The password change doesn't affect the task and it will continue to work. If the task accesses external resources, however, you'll need to update the password associated with just one task on each computer that uses those credentials.

Tasks can have many properties associated with them, including:

- Triggers that specify the circumstances under which a task begins and ends
- Actions that define what the task does when it is started
- Conditions that qualify the conditions under which a task is started or stopped
- Settings that affect the behavior of the task
- History that shows events generated while a task is running or attempting to run

Unlike task scheduling for earlier versions of Windows, you can specify more than one trigger and more than one action. This means each scheduled task can run multiple programs, utilities, or scripts, and can be configured to run in a number of ways, including:

- At a specific time and date, such as at 5:45 P.M. on October 25, 2015
- At a specified interval, such as every Monday, Wednesday, and Friday at 5:45 P.M.
- When a specific system event occurs, such as when someone logs on to the system
- Any combination of the previous examples, as well as other ways not listed

Task triggers deserve special attention because they don't always work as you might expect and include tasks triggered by the following events:

- **System Start** If you schedule a task to run at startup, Task Scheduler runs the task as a noninteractive process whenever the computer is started. The task will continue to run until it finishes, is terminated, or the system is shut down. Keep in mind that only the owner of the task or an administrator can terminate running tasks.
- **System Logon** If you schedule a task to run when a user logs on, you can configure the task to run whenever anyone logs on (regardless of who configured the task to run) or only when a specified user logs on. Task Scheduler will then continue to run until the task finishes, is terminated, or the user logs off. Logon tasks can run interactively or noninteractively, depending on how they are configured.

TIP If a user configures an interactive task using his or her logon and someone else logs on, the task runs with the original user's permissions and may not terminate when the other user logs off (because it is owned by someone else and the current user may or may not have appropriate permissions to terminate the task). Further, with Fast User Switching, logon tasks do not run when you switch users. Logon tasks only run when someone logs on while all users are logged off.

- **System Idle** If you schedule a task to run when the system is idle, Task Scheduler runs the task whenever no user activity occurs for a specified amount of time. For example, you might create a task that runs only when the system has been idle for five minutes. Keep in mind, however, that subsequent user activity will not terminate the task. The task will continue to run until it finishes or is terminated.
- **Windows Event** If you schedule a task to run based on a Windows event, Task Scheduler runs the task whenever Windows writes an event with the identifier you specify to a

particular event log. In the basic configuration, you can specify a single event identifier, an optional source, and a single event log to monitor. With a custom configuration, you can define an event filter that works like the filters we've discussed in previous chapters and allows you to monitor multiple logs, multiple sources, or both for multiple types of events.

- **User Session** If you schedule a task to run when a user establishes a Terminal Services user session, Task Scheduler runs the task when a user creates a user session by connecting from the local computer or from a remote computer. You can also schedule a task to run when a user ends a Terminal Services user session from the local computer or from a remote computer.

Although you can define multiple actions for a task, you can create a command-line script that runs multiple commands, programs, and utilities and performs other necessary tasks as well. Here, you'll want the script to run with specific user or administrator credentials to ensure that the script has the necessary permissions and access rights. The script should also configure whatever user settings are necessary to ensure that everything it does is under its control and that domain user settings, such as drive mappings, are available as necessary.

> *REAL WORLD* When you configure tasks to run, you can specify the user account and logon password to use when the task runs. With recurring tasks, this tactic can lead to problems, especially if permissions or passwords change—and they inevitably do. If account permissions or passwords change and the task works with remote resources, you'll need to edit the properties of at least one task that uses those credentials and supply the new credentials for the account.

Monitoring Scheduled Tasks

Task Scheduler doesn't verify the information you provide or the availability of programs, commands, or utilities. If you don't specify the correct information, the task simply won't run or will generate errors when it does run. One way to check tasks is to view their status and last result in the Task Scheduler. This information pertains to the last time the scheduled task ran. If the Last Run Result is an error, you'll need to resolve the referenced problem so that the task can run normally. Check a task's properties by clicking its entry in Task Scheduler.

A task that is listed as Running might not in fact be running but instead might be a nonresponsive or runaway process. You can check for nonresponsive or runaway processes using Last Run Time, which tells you when the task was started. If a task has been running for more than a day, a problem usually needs to be corrected. A script might be waiting for input, it might have problems reading or writing files, or it might simply be a runaway task that needs to be stopped. To stop the task, right-click it in Task Scheduler and then select End.

The Last Run Result won't tell you, however, if there were problems running tasks prior to the last run time. To dig deep and get a better understanding of how tasks are running, you should periodically check the Task Scheduler operational log. In Event Viewer, this log is stored under Applications And Services Logs\Microsoft\Windows\TaskScheduler\Operational. If you examine the task scheduler log, you'll find the following information:

- Entries that record when the Task Scheduler Service was started and when the service exited (was stopped).
- Entries that record when tasks are started, when they finished running, and the exit error or result code. An exit error or result code of zero (0) means that the task executed normally. Any other exit or result code indicates an error may have occurred.
- Entries that record warnings and errors that occurred when Task Scheduler tried to start an event.

TIP The command-line name for the Task Scheduler operational log is Microsoft-Windows-TaskScheduler/Operational. By default the log is stored in the %SystemRoot%\System32\Winevt\Logs folder with the name Microsoft-Windows-TaskScheduler/Operational.evtx. Using the techniques discussed in Chapter 7, "Event Logging, Tracking, and Monitoring," you can manipulate this log as you would any other log.

Task Scheduler has custom filtered views of the operational log. When you select the Task Scheduler node in Task Scheduler, the main pane has several panels, including Task Status and Active Tasks. Using the options on the Task Status panel, you can view a summary status for all scheduled tasks in the last hour, last 24 hours, last 7 days, or last 30 days. On the Active Tasks panel, you can view summary information about running tasks. If you double-click a running task, you'll access the task definition within the Task Scheduler Library.

When you select a task definition in the Task Scheduler and then click the History tab, you'll see a filtered view of the Task Scheduler operational log with events specific to that task. If you select a specific event in turn, you'll see detailed information about the event in the lower pane.

You may also want a more detailed understanding of what happens when scripts run. To do this, you may want to record the output of commands and utilities in a separate log file, thereby giving you the opportunity to determine that those commands and utilities produced the expected results. You can write command output to a named file by redirecting standard output and standard error. In the following examples, the output of the DEFRAG command is appended to Stat-log.txt and any DEFRAG errors are written to this file as well:

```
defrag c: >> c:\logs\stat-log.txt 2>&1
defrag d: >> c:\logs\stat-log.txt 2>&1
```

CAUTION If you are working with a directory, as shown in these examples, the directory must already exist. It will not be created for you, and any errors resulting from the lack of a directory will not be written to the log file.

REAL WORLD Writing command output to a log file won't help you resolve every problem that can occur, but it goes a long way toward ensuring that scheduled tasks run as expected. If you are trying to troubleshoot problems, keep in mind that tasks can fail to run for many reasons, some of which are beyond your control. For example, scheduled tasks won't run if the system is shut down when the task is scheduled to run. If you want to ensure that a task runs even if the scheduled start time is missed, you must access the Settings tab in the task's Properties dialog box and select Run Task As Soon As Possible After A Scheduled Start Is Missed. With this feature enabled, Windows runs a scheduled task as soon as possible if you missed a task because the system was shut down.

Scheduling Tasks with Task Scheduler

You can use Task Scheduler to create basic and advanced tasks. Basic tasks include triggers and actions that are meant to help you quickly schedule a common task. Advanced tasks include triggers, actions, conditions, and settings that are meant to be used by advanced users or administrators.

Creating Basic Tasks

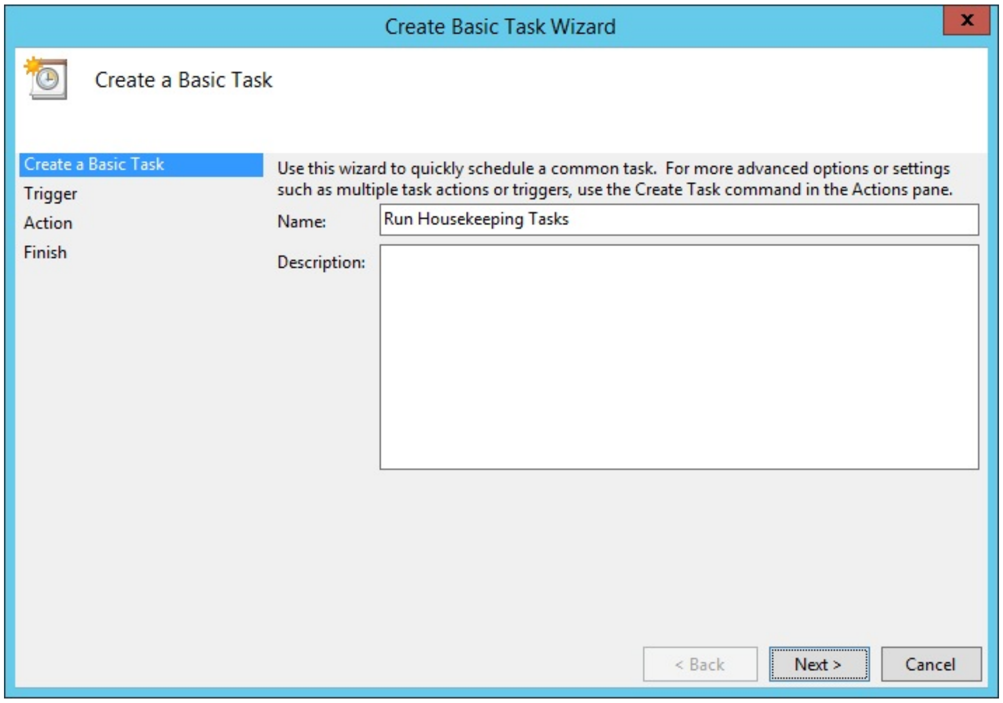
Basic tasks have many default settings and are easy to create. By default, basic tasks you create run under your logon account and will run only when you are logged on. These tasks run with standard user privileges rather than the highest possible privileges and are configured for compatibility with Windows 8.1, Windows Server 2012, and Windows Server 2012 R2. These tasks only start if the computer is on AC power and stop when the computer switches to battery power. Additionally, basic tasks stop when they've been running for longer than three days.

You can create a basic task by completing these steps:

1. Open Task Scheduler by opening Server Manager, clicking Tools, and then clicking Task Scheduler. You are connected to the local computer by default. If you want to view logs on a remote computer, right-click the Task Scheduler entry in the console tree (left pane) and

then select Connect To Another Computer. Then, in the Select Computer dialog box, enter the name or IP address of the computer that you want to access and click OK.

2. You can create tasks at any level within the Task Scheduler Library. Right-click the node you want the task to be stored in and then select Create Basic Task. This starts the Create Basic Task Wizard.
3. On the Create A Basic Task page, shown in Figure 9-2, type a name and description of the task. The name should be short but descriptive so that you can quickly determine what the task does. The optional description can provide a detailed explanation of the task's purpose. Click Next.



Use this wizard to quickly schedule a common task. For more advanced options or settings such as multiple task actions or triggers, use the Create Task command in the Actions pane.

Name: Run Housekeeping Tasks

Description:

FIGURE 9-2 Create a basic task using the Create Basic Task Wizard.

4. On the Task Trigger page, select a run schedule for the task. You can schedule tasks to run one time, periodically (daily, weekly, or monthly), when a specific event occurs when the computer starts, or when the task's user logs on. Click Next. The next page you see depends on when the task is scheduled to run.
5. If you've selected a daily running task, the wizard displays the Daily page. Configure the task using the following fields and then click

Next:

Start Use the Start options to set a start date and time.

Synchronize Across Time Zones Select this option to schedule the task according to Greenwich Mean Time (GMT) instead of local time. GMT is the time at 0 degrees meridian and it is the time zone for Greenwich, London, England.

Recur Allows you to run the task every day, every other day, or every nth day, beginning with the start date you set. For example, if you want the task to run every other day, you'd set the Recur Every: . . . Days text box to 2 days.

6. If you've selected a weekly running task, the wizard displays the Weekly page. Configure the task using the following fields and then click Next:

Start Use the Start options to set a start date and time.

Synchronize Across Time Zones Select this option to schedule the task according to GMT instead of local time.

Recur Allows you to run the task every week, every other week, or every nth week.

Days of the Week Sets the day(s) of the week when the task runs, such as on Tuesday, or on Tuesday and Friday.

7. If you've selected a monthly running task, the wizard displays the Monthly page. Configure the task using the following fields and then click Next:

Start Use the Start options to set a start date and time.

Synchronize Across Time Zones Select this option to schedule the task according to GMT instead of local time.

Months Use this selection list to choose which months the task runs. You can select all months, or one or more individual months.

Days Sets the day(s) of the month the task runs. For example, if you select 2 and 8, the task runs on the second and eighth days of the month.

On Sets the task to run on the nth occurrence of a day in a month, such as the second Monday, or the first and third Tuesday of every month.

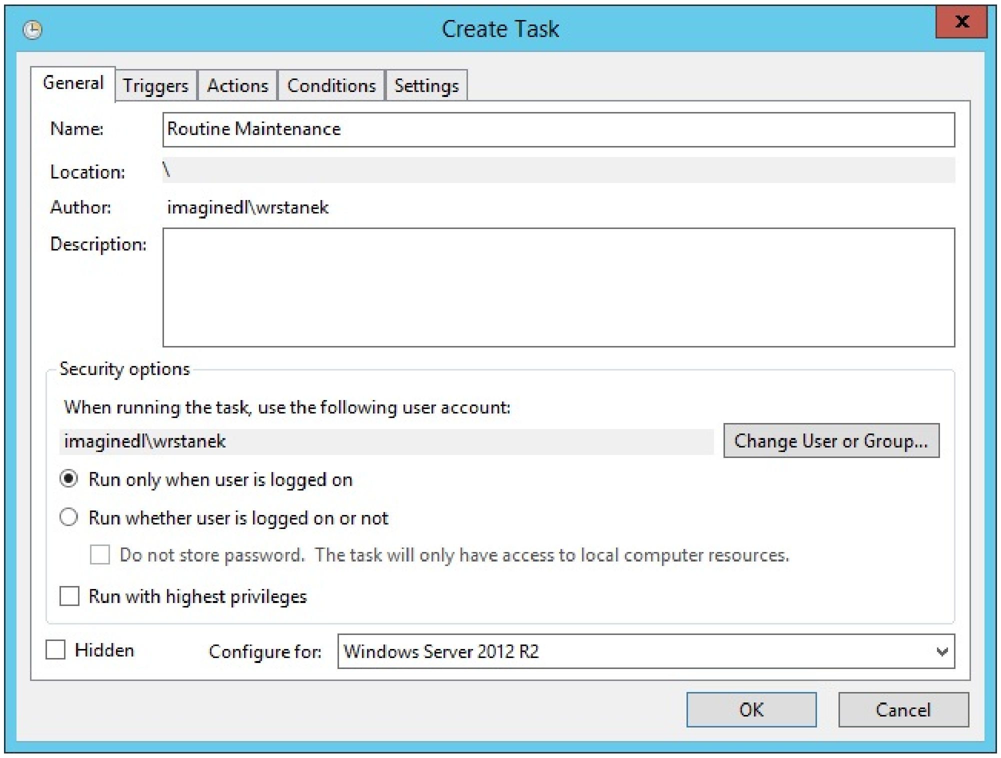
8. If you've selected One Time for running the task, the wizard displays the One Time page. Use the Start options to set a start date and time. Click Next.
9. If you've selected When A Specific Event Is Logged, the wizard displays the When A Specific Event Is Logged page. You'll need to select the event log to monitor and the specific event source, event ID, or both. Click Next.
10. If you've selected When I Log On or When The Computer Starts, clicking Next takes you immediately to the Action page.
11. On the Action page, specify the task to perform. You can start a program, send an e-mail, or display a message. The next page you see depends on the action you selected. Click Next.
12. If you've selected Start A Program, the wizard displays the Start A Program page. Click Browse to display the Open dialog box and then select the program or script to run. Click Next.
13. If you've selected Send An E-mail, the wizard displays the Send An E-mail page. You can then configure the automated e-mail to send by completing the From, To, Subject, and Text fields of the e-mail message. In the SMTP Server text box, enter the fully qualified domain name of the mail server through which you will send your message. Click Next.
14. If you've selected Display A Message, the wizard displays the Display A Message page. You can then configure the message to display on the desktop when the task is started. Enter the title and text of your message in the text boxes provided. Click Next.
15. On the Summary page, review the task details and then click the Finish button. Once the task is created, you can modify the basic task's default settings by accessing its Properties dialog box.

Creating Advanced Tasks

With advanced tasks, you configure the task settings directly in a dialog box similar to the standard properties dialog box for tasks. Advanced tasks have the same default settings initially as basic tasks.

You can create an advanced task by completing these steps:

1. Open Task Scheduler by opening Server Manager, clicking Tools, and then clicking Task Scheduler. You are connected to the local computer by default. If you want to view logs on a remote computer, right-click the Task Scheduler entry in the console tree (left pane) and then select Connect To Another Computer. Then, in the Select Computer dialog box, enter the name or IP address of the computer that you want to access and then click OK.
2. You can create tasks at any level within the Task Scheduler Library. Right-click the node you want the task to be stored in and then select Create Task. This opens the Create Task dialog box.
3. On the General tab, shown in Figure 9-3, type a name and description of the task. The name should be short but descriptive so that you can quickly determine what the task does. The optional description can provide a detailed explanation of the task's purpose.



Name: Routine Maintenance

Location: \

Author: imaginedl\wrstanek

Description:

Security options

When running the task, use the following user account:

imaginedl\wrstanek

[✓ Run only when user is logged on]

Run whether user is logged on or not

☐ Do not store password. The task will only have access to local computer resources.

☐ Run with highest privileges

☐ Hidden Configure for: Windows Server 2012 R2

FIGURE 9-3 Create an advanced task using the Create Task dialog box.

4. By default, the task runs under your user account. To run the task under a different user account, click Change User Or Group and then use the dialog box provided to select the user or group under which the task should run.
5. By default, the task runs only when you or another specified user is logged on. If you want to run the task regardless of whether you are logged on, select Run Whether User Is Logged On Or Not. With this setting, Task Scheduler stores your credentials or the specified user's credentials.

NOTE If don't want Task Scheduler to store the password associated with these credentials, select the Do Not Store Password check box. Keep in mind, however, that this setting will allow the task to be used to access local computer resources but will prevent the task from being used to access remote computer resources.

6. By default, the task runs with standard user privileges rather than the highest possible privileges associated with your account or the specified user's account. If the task requires administrative privileges

to run, select the Run With Highest Privileges check box.

7. By default, Task Scheduler always displays the task. To identify the task as a hidden task, select the Hidden check box. This hides the task from the default view and requires users to elect to display hidden tasks if they want to view the task.
8. By default, the task is created for compatibility with earlier operating systems. To ensure the task runs more efficiently, you may want to select the current operating system in the Configure For list. However, don’t do this if you may export the task and import to an earlier version of Windows.
9. On the Triggers tab, create and manage triggers using the options provided. Using triggers, you can schedule tasks to run one time, periodically (daily, weekly, or monthly), or when a specific event occurs, such as when the computer starts or when the task’s user logs on. To create a trigger, click New, use the options provided to configure the trigger, and then click OK. You can define multiple triggers.
10. On the Actions tab, create and manage actions using the options provided. You can start a program, send an e-mail, or display a message. To create an action, click New, use the options provided to configure the action, and then click OK. You can define multiple actions.
11. On the Conditions tab, specify any limiting conditions for starting or stopping the task.
12. On the Settings tab, choose any additional optional settings for the task.
13. Click OK to create the task. Once the task is created, you can modify the task’s settings by accessing its Properties dialog box.

Managing Task Properties

You can access and work with the tasks configured on your computer through Task Scheduler. Select a task to view its properties using the tabs provided in the lower portion of the main pane. Note the task status, last run time, and last run result. On a task’s History tab, you’ll see a detailed history of the task. Use the history information to help you resolve problems with the task. If you want to change a task's properties, you can double-click the task and then use the Properties dialog box to make the necessary changes.

Enabling and Disabling Tasks

You can enable or disable tasks as needed, depending on your preference. If you temporarily don't want to use a task, you can disable it. When you are ready to use the task again, you can enable it. By enabling and disabling tasks rather than deleting them, you save the time involved in reconfiguring task settings. To disable a task, right-click the task and then select Disable. To enable a task, right-click the task and then select Enable.

Copying Tasks to Other Computers

The import and export options in Task Scheduler make it very easy to copy tasks from one computer to another. You can copy a task to another computer by following these steps:

1. Open Task Scheduler by opening Server Manager, clicking Tools, and then clicking Task Scheduler.
2. If you aren't logged on to the source computer with the task you want to copy, right-click the Task Scheduler node and then select Connect To Another Computer. Use the Connect To Another Computer to connect to the source computer.
3. Right-click the task you want to copy and then select Export Task. In the Save As dialog box, select a save location and filename for the task's XML configuration file, and then click Save.
4. Right-click the Task Scheduler node and then select Connect To Another Computer. Use the Select Computer dialog box to connect to the destination computer.
5. Right-click the Task Scheduler node and then select Import Task. In the Open dialog box, access the save location of the task's XML

configuration file. Click the configuration file to select it and then click Open.

6. Task Scheduler reads the configuration file and then opens the Create Task dialog box with the task's settings preconfigured. Using the Create Task dialog box, modify the task's original settings as necessary for the destination computer. Click OK.

Running Tasks Immediately

You don't have to wait for the scheduled time to run a task. To run a task at any time, access the Task Scheduler, right-click the task you want to run, and then select Run.

Removing Unwanted Tasks

If you no longer need a task, you can delete it permanently by accessing the Scheduled Tasks folder, right-clicking the task you want to delete, and then selecting Delete.

Scheduling Tasks with Schtasks

With Schtasks, you can perform the same task scheduling operations as with the Task Scheduler wizard. Any tasks you create using Schtasks are displayed as scheduled tasks in the Task Scheduler and can be managed from the command line or from the graphical user interface (GUI).

Schtasks has several different sets of subcommands and is one of the more complex utilities available at the command line. The sections that follow discuss each of the following subcommands:

- **Schtasks /Create** Use to create scheduled tasks.
- **Schtasks /Change** Use when you want to change the properties of existing tasks.
- **Schtasks /Delete** Use to remove scheduled tasks that are no

longer wanted.

- **Schtasks /End** Use to stop a running task.
- **Schtasks /Query** Use to display scheduled tasks on the local or named computer.
- **Schtasks /Run** Use to start a scheduled task immediately.
- **Schtasks /ShowSid** Use to display the security identifier of the user associated with the task.

Creating Scheduled Tasks with Schtasks /Create

With Schtasks /Create, you can create one-time-only tasks, recurring tasks, and tasks that run based on specific system events, such as logon and startup. The basic syntax for defining these types of tasks is as follows:

```
schtasks /create /tn TaskName /tr TaskToRun /sc ScheduleType
[/mo Modifier]
```

where *TaskName* sets the task name string, *TaskToRun* specifies the file path to the program, command-line utility, or script that you want to run, *ScheduleType* specifies the run schedule, and *Modifier* is an optional value that modifies the run schedule based on the schedule type. Any tasks you create using this syntax are created on the local computer and use your user permissions. Further, if you don’t provide your account password, you are prompted for your user password when you create tasks.

Valid values for *ScheduleType* are shown in Table 9-1. Note the usage and modifiers that the various schedule types accept. I’ll discuss each schedule type and modifier in detail later in the chapter. Note the following as well:

- You can enter days of the week in a comma-separated list, such as Mon, Wed, Fri, or with a hyphen to specify a sequence of days, such as Mon-Fri for Monday through Friday.
- You can enter months of the year in a comma-separated list,

such as Jan, Mar, Jun, or with a hyphen to specify a sequence of months, such as Jan-Jun for January through June.

- With week of the month, you can only enter one value, such as FIRST or LAST.

TABLE 9-1 Schedule Types for Schtasks /Create

SCHEDULE TYPE	DESCRIPTION	MODIFIER VALUES
MINUTE	Task runs at a specified interval in minutes. By default, tasks run once a minute.	/mo 1-1439; the number of minutes between each run of the task. The default modifier is 1.
HOURLY	Task runs at a specified interval in hours. By default, tasks run once an hour.	/mo 1-23; the number of hours between each run of the task. The default modifier is 1.
DAILY	Task runs every day or every *n* days. By default, tasks run once a day.	/mo 1-365; the number of days between each run of the task. The default modifier is 1.
WEEKLY	Task runs every week or every *n* weeks, on designated days. By default, tasks run once a week on Mondays.	/mo 1-52; the number of weeks between each run of the task. Optionally, use /d to specify the days of the week to run. Use MON, TUE, WED, THU, FRI, SAT, and SUN to specify days. Use * for every day of the week.
MONTHLY	Task runs every month or every *n* months on designated days. By default, tasks run the first day of every month.	/mo 1-12; the number of months between each run of the task. Optionally, use /d MON-SUN; sets day of the week to run during the month. Use * to have the task run every day.
	Second monthly variant for specific day of month. Use *mo and* m, or *m and* d.	

/mo LASTDAY; last day of month.

/m JAN, FEB, . . ., DEC; sets the month(s).

/d 1-31; day of month.

Third monthly variant for specific week of the month.

/mo FIRST | SECOND | THIRD | FOURTH | LAST; sets week of month.

/d MON-SUN; sets day of week.

/m JAN, FEB, . . ., DEC; sets month(s).

ONCE

Task runs once at a specified date and time.

—

ONEVENT

Tasks run when a specified event or events occur in a specified event log.

/mo *XPathString* where *XPathString* is the XPath event query string that identifies the event on which the scheduled task is triggered.

ONSTART

Task runs whenever the system starts.

—

ONLOGON

Task runs whenever a user logs on.

—

ONIDLE

Task runs whenever the system is idle for a specified period of time.

/i 1-999; the number of minutes the system has to be idle before the task starts.

To see how you can use Schtasks /Create, consider the following examples: Task runs once immediately and then doesn’t run again: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc once` Task runs when the system starts:

`schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc onstart` Task runs whenever the system is idle for more than 10 minutes: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc onidle /i 10`

Task runs every 15 minutes on the local computer: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc minute /mo 15`

Task runs every five hours on the local computer: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc hourly /mo 5`

Task runs every two days on the local computer: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc daily /mo 2`

Task runs every two weeks on Monday (the default run day): `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc weekly mo 2`

Task runs every week on Monday and Friday: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc weekly d mon,fri` Task runs on the first day of every month: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc monthly` Task runs on the fifth day of every other month: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc monthly  mo 2 /d 5`

Task runs the last day of every month: `schtasks /create tn "SysChecks" tr c:\scripts\sysch.bat sc monthly.mo lastday` Task runs the first Monday of April, August, and December: `schtasks /create tn "SysChecks" tr c:\scripts\sysch.bat /sc monthly mo first d mon /m apr,aug,dec` When the path of the specified task includes a space, enclose the file path in double quotation marks as shown in the following example: `schtasks /create tn "SysChecks" tr "c:\My Scripts\sch.bat" /sc onstart` If you do not enclose the file path in quotation marks, an error will occur when Schtasks attempts to run the task. Further, if you want to pass arguments to a program, utility, or script, simply follow the Task To Run file path with the arguments you want to use. Any argument that contains spaces should be enclosed in quotation marks so that it is properly interpreted as a single argument rather than multiple arguments. Here are examples: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat 1 Y LAST /sc onstart`

```
schtasks /create tn "SysChecks" tr "c:\My Scripts\sch.bat" Y N
/sc onstart

schtasks /create tn "SysChecks" tr "c:\My Scripts\sch.bat" "Full Checks"
```

You can also schedule tasks for remote computers as well as tasks that should run with different user permissions. The key detail to remember when scheduling tasks on remote computers is that the computer you are using should be in the same domain as the remote computer or in a domain that the remote computer trusts. To do this, you must use the expanded syntax, which includes the following parameters: `s Computer u [Domain\]User [/p Password]`

where *Computer* is the remote computer name or IP address, *Domain* is the optional domain name in which the user account is located, *User* is the name of the user account whose permissions you want to use, and *Password* is the optional password for the user account. If you don't specify the domain, the current domain is assumed. If you don't provide the account password, you are prompted for the password.

To see how you can add the computer and user information to the syntax, consider the following examples: Use the account imaginedlands\wrstanek when creating the task on the local computer: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc onstart u imaginedlands\wrstanek p RoverSays` Set the remote computer as mailer01 and the account to use as adatum\wrstanek: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat /sc onstart /s mailer01 u imaginedlands\wrstanek p RoverSays` Using the /Ru and *Rp parameters, you can specify the credentials for the user account under which the task should run. If you want a task to run only when a specific* *user is logged on, use the optional* It parameter, which specifies that the task should run interactively and only when the user who owns the task is logged on. With tasks that work only with local resources, you can use the /Np parameter to specify that no password should be saved with a user's credentials. When you don't allow Task Scheduler to save a password, the task only has access to local resources and runs non-interactively as the specified user.

Tasks run with standard user privileges by default. If you want a task to run with the highest privileges of the specified user, such as may be necessary for administrative tasks, you can set the /Rl parameter to Highest rather than the default value Limited.

To see how you can add alternate credentials and privileges to the syntax, consider the following examples: Configure the task to run using the credentials of imaginedlands\thomasv: `schtasks create tn "CleanUp" tr c:\scripts\cleanup.bat sc onlogon ru imaginedlands homasv rp DingoE`

Set the remote computer as server18, the account to use for creating the task as imaginedlands\wrstanek, and configure the task to run using the credentials of imaginedlands\thomasv: `schtasks create tn "CleanUp" tr c:\scripts\cleanup.bat sc onlogon /s server18 u imaginedlands\wrstanek p RoverSays ru imaginedlands homasv rp DingoE`

Run the task without saving the user password on server18 using the account imaginedlands\wrstanek: `schtasks create tn "CleanUp" tr c:\scripts\cleanup.bat sc onlogon /s server18 u imaginedlands\wrstanek np` Run the task with the highest privileges: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc onlogon rl highest` Finally, if desired, you can add specific start times and dates, as well as end times and dates, using the following values:

- /st *StartTime*, where *StartTime* is in 24-hour clock format (HH:MM), such as 15:00 for 3:00 P.M. This parameter is required with /Sc ONCE.
- /et *EndTime*, where *EndTime* is in 24-hour clock format (HH:MM), such as 15:00 for 3:00 P.M. This parameter is not

applicable for schedule types ONSTART, ONLOGON, ONIDLE, and ONEVENT.

- /du *Duration*, where *Duration* is the number of hours and minutes to run, in the form HHHH:MM. This parameter is not applicable with /Et and for schedule types ONSTART, ONLOGON, ONIDLE, and ONEVENT.
- /sd *StartDate*, where *StartDate* is the start date using the default system format for dates, such as MM/DD/YYYY. This parameter is not applicable for schedule types ONCE, ONSTART, ONLOGON, ONIDLE, and ONEVENT.
- /ed *EndDate*, where *EndDate* is the end date using the default system format for dates, such as MM/DD/YYYY. This parameter is not applicable for schedule types ONCE, ONSTART, ONLOGON, ONIDLE, and ONEVENT.

TIP If you specify an end date or time, you can also specify the / Z parameter, which tells Task Scheduler to delete the task upon completion of its schedule.

To see how you can use specific start times and dates, as well as end times and dates, consider the following examples: Start the hourly task at midnight: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc hourly st 00:00`

Start the hourly task at 3:00 A.M. and stop it at 7:00 A.M.: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc hourly st 03:00 /et 07:00`

Start the weekly task at 3:00 A.M. on February 20, 2015: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc weekly st 03:00 /sd 02/20/2015`

Start the weekly task at 3:00 A.M. on February 20, 2015 and end at 2:59 A.M. on March 15, 2015: `schtasks /create tn "SysChecks" tr c:\scripts\sch.bat sc weekly st 03:00 /sd 02/20/2015 et 02:59 ed 03/15/2015`

NOTE Date and time formats are determined by the Regional And

Language Options settings used by the computer. In these examples, the date format preference is English (United States).

Creating Scheduled Tasks Triggered by Windows Events

With Schtasks /Create, you can create scheduled tasks that start when the operating system or a Windows component writes specific events or types of events to one of the event logs. The basic syntax for event-triggered tasks is as follows: `schtasks create tn` *TaskName* `/tr` *TaskToRun*
`/sc` *ONEVENT*
`/ec` *LogName* `/MO` *EventString* where *TaskName* sets the task name string, *TaskToRun* specifies the file path to the program, command-line utility, or script that you want to run, *LogName* sets the name of the event log to monitor, and *EventString* sets the XPath event query string that identifies the event or events on which the scheduled task is triggered.

Using /Sc ONEVENT with the schedule task definition is what schedules the task to trigger based on an event. With the *Ec parameter, you specify the command-line compatible name of the event log to monitor. You can use the Wevtutil el command to list all available event logs on a computer in a command-line compatible format. With the* Mo parameter, you define the query by specifying the XPath event query string.

You don’t have to try to create query strings from scratch. Instead, use Event Viewer to create a filter that identifies the exact event or events you want to monitor. Then, copy the related XPath query to Notepad or any standard text editor. Once you’ve copied the query to Notepad, you should save the query so that you have the original settings and then try to extract the necessary event query string. Generally, the event query string you need is the text between the begin <Select> tag and the end </Select> tag within the Query element. Consider the following example:

```
<QueryList>
  <Query Id="0" Path="Application">
    <Select Path="Application">*[System[(Level=1  or Level=2 or
Level=3)]]</Select>
  </Query>
</QueryList>
```

In this example, the event query string is: `*[System[(Level=1 or Level=2 or Level=3)]]`

This query string creates a filter that looks for critical, warning, and error events in the applicable event log. The following example uses this query to create a scheduled task called Track Application Issues to run Event Viewer whenever critical, warning, and error events are written to the applicable event log: `schtasks create tn "Track Application Issues" tr wevtvwr.msc sc` *ONEVENT*
`/ec` *Application* `/`*MO* `"*[System[(Level=1  or Level=2 or Level=3)]]"`

Note that because the task name and the query string contain spaces, they are enclosed in double quotation marks. While this type of query string is probably too broad, it is a good example of what a query string looks like. A better query is one that identifies a specific event by its event identifier. The event query string syntax for specifying a single event identifier is: `*[System[(EventID=`*EventNumber*`)]]`

where *EventNumber* is the identifier number of the event to monitor. In the following example, you create a scheduled task that runs whenever event id 3210 is written to the System log: `schtasks create tn "Computer Authentication Issues" /tr wevtvwr.msc`
`/sc` *ONEVENT* `/`*ec System* `/`*MO* `"*[System[(EventID=3210)]]"`

> *REAL WORLD* Events with Event ID 3210 are written to the System log when a computer is unable to authenticate in the domain. This error can occur because the computer's password needs to be reset. This error can also occur if another computer on the network has the same name.

By extending the query string with or statements, you can enter multiple event identifiers. The extended syntax looks like this: `*[System[(EventID=`*EventNumber* `or EventID=`*EventNumber* `or ...)]]`

In the following example, you create a scheduled task that runs whenever event id 3210 or event id 5722 are written to the System log: `schtasks create tn "Computer Authentication Issues" /tr wevtvwr.msc sc ONEVENT ec System /MO "*[System[(EventID=3210 or EventID=5722)]]"`

> *REAL WORLD* Events with Event ID 5722 are written to the System log when a computer is denied access to a resource. This error can

occur because the computer account has been disabled or deleted.

Changing Scheduled Tasks with Schtasks /Change

You use Schtasks /Change to change key parameters associated with scheduled tasks. The basic syntax of Schtasks /Change is `schtasks change tn TaskName ParametersToChange` where *TaskName* is the name of the task you want to change and *ParametersToChange* are the parameters you want to change. Parameters you can work with include the following:

- /ru *Domain*\User changes the user under which the task runs, such as /ru imaginedlands\wrstanek. For the system account, valid values are "", "NT AUTHORITY\SYSTEM", or "SYSTEM". For tasks configured to work with Windows 8.1, Windows Server 2012, or Windows Server 2012 R2, you can also use "NT AUTHORITY\LOCALSERVICE" and "NT AUTHORITY\NETWORKSERVICE".
- /rp *Password* sets the password for the previously specified or newly designated runas user account. To prompt for the password, use "*" or leave blank. This password is ignored for the system account. You can only set the runas user password when you also specify the runas user account.
- /tr *TaskToRun* changes the program, command-line utility, or script that is run for the named task.
- /st *StartTime* sets the start time for minute or hourly tasks. *StartTime* is in 24-hour clock format (HH:MM), such as 15:00 for 3:00 P.M. This parameter is required if /Sc ONCE was specified during task schedule creation.
- /ri *Interval* sets the repetition interval in minutes, with a valid range of 1–599,940 minutes. This parameter is not applicable for schedule types MINUTE, HOURLY, ONSTART, ONLOGON, ONIDLE, and ONEVENT.. If either *Et or* Du is specified, the repetition interval defaults to 10 minutes.

- /et *EndTime* sets the end time for minute or hourly tasks. *EndTime* is in 24-hour clock format (HH:MM), such as 15:00 for 3:00 P.M. This parameter is not applicable for schedule types ONSTART, ONLOGON, ONIDLE, and ONEVENT.
- /du *Duration* sets the number of hours and minutes to run the task, in the form HHHH:MM. This parameter is not applicable with /Et and for schedule types ONSTART, ONLOGON, ONIDLE, and ONEVENT.
- /sd *StartDate* sets the start date for the task using the default system format for dates, such as MM/DD/YYYY. This parameter is not applicable for schedule types ONCE, ONSTART, ONLOGON, ONIDLE, and ONEVENT.
- /ed *EndDate* sets the end date for the task using the default system format for dates, such as MM/DD/YYYY. This parameter is not applicable for schedule types ONCE, ONSTART, ONLOGON, ONIDLE, and ONEVENT.
- /k specifies that the task should not be started again when the end time or duration interval is reached, but it doesn't stop the task if it is already running. (The current run will be the last one.) This parameter is not applicable for schedule types ONSTART, ONLOGON, ONIDLE, and ONEVENT. Either *Et or* Du must be specified.
- /it specifies that the task should run only when the user who owns the task is logged on.
- /rl Level sets the run level for the task as Limited for standard user privileges or Highest for the highest possible privileges of the runas user.
- /delay *DelayTime* sets the time delay for running a task after it is triggered. *DelayTime* is set in the form mmmm:ss and can only be set for schedule types ONSTART, ONLOGON, and ONEVENT.
- /enable enables the schedule task, allowing the task to run according to its schedule.

- /disable disables the schedule task, preventing the task from running.
- /z marks the task for deletion after its final, scheduled run.

To see how you can change tasks, consider the following examples: Change the script that is run:

`schtasks /change tn "SysChecks" tr c:\scripts\systemchecks.bat` Change the runas user and password: `schtasks /change tn "SysChecks" ru adatum\hthomas /rp gophers` Change task to start weekly at 7:00 A.M. on March 1, 2015 and end at 6:59 A.M. on March 30, 2015: `schtasks /change tn "SysChecks" st 07:00 /sd 03/01/2009 et 06:59 ed 03/30/2015`

> *NOTE* As mentioned previously, date and time formats are determined by the Regional And Language Options settings used by the computer. Here, the date format is English (United States).

When you change a task, Schtasks displays a message that states whether the changes succeeded or failed, such as: `SUCCESS: The parameters of the scheduled task "SysChecks" have been changed.`

If you are working with a remote computer or aren't logged in with a user account that has permission to change the task, you can specify the computer and account information as necessary. The syntax is `schtasks change tn` *TaskName* `/s` *Computer* `/u [`*Domain*`\]`*User*
`[/p` *Password*`]`

where *Computer* is the remote computer name or IP address, *Domain* is the optional domain name in which the user account is located, *User* is the name of the user account whose permissions you want to use, and *Password* is the optional password for the user account. If you don't specify the domain, the current domain is assumed. If you don't provide the account password, you are prompted for the password.

In the following example, the remote computer is mailer01 and the user account that has authority to change the SysChecks task is wrstanek's Imaginedlands domain account: `schtasks /change tn "SysChecks" tr c:\scripts\systemchecks.bat` `s mailer01 u imaginedlands\wrstanek` Because a password isn't specified, Schtasks will prompt for one.

You can also quickly enable or disable tasks by name. Use the following syntax to enable tasks: `schtasks change tn` *TaskName* `/enable` Use this syntax to disable tasks:

`schtasks change tn` *TaskName* `/disable` where *TaskName* is the name of the task you want to enable or disable, such as: `schtasks /change tn "SysChecks" disable`

Querying for Configured Tasks with Schtasks /Query

You can quickly determine what tasks are configured on a computer by typing **schtasks /query** at the command prompt and, as necessary for a remote computer, you can specify the computer and the account information needed to access the computer using the following form: `schtasks query s` *Computer* `/u [`*Domain*`\]`*User* `[/p` *Password*`]`

where *Computer* is the remote computer name or IP address, *Domain* is the optional domain name in which the user account is located, *User* is the name of the user account with appropriate access permissions on the remote computer, and *Password* is the optional password for the designated user account.

In the following example, the remote computer is mailer01 and the user account is wrstanek's imaginedlands domain account: `schtasks /query s mailer01 u imaginedlands\wrstanek` Because a password isn't specified, Schtasks will prompt for one.

The basic output of Schtasks /Query is in table format and provides TaskName, Next Run Time, and Status columns. You can also format the output as a list or lines of comma-separated values using *Fo List or* Fo Csv, respectively. The list output works best with the /V (verbose) parameter, which provides complete details on all task properties and which you can use as shown in the following example: `schtasks query s mailer01 u imaginedlands\wrstanek fo list /v` Another useful parameter is

/Nh, which specifies that table-formatted or CSV-formatted output should not have headings.

TIP You may wonder why you'd want to use the various formats. It's a good question. I recommend using the verbose list format (/Fo List /V) when you want to see all details about tasks configured on a system and when you are troubleshooting, and I recommend using comma-separated values when you want to store the output in a file that may later be exported to a spreadsheet or flat-file database. Remember that you can redirect the output of Schtasks to a file using output redirection (> or >>).

Creating Tasks Using XML Configuration Files

The /Xml parameter is one of the parameters we haven't talked about for Schtasks *Create. When you use this parameter with Schtasks* Create, you can specify the XML configuration file that defines the new task you are creating. The basic syntax is: `schtasks create tn` *`TaskName`* `/xml` *`XmlFile`* where *TaskName* is the name of the task to create and *XmlFile* specifies the name or full file path to the XML configuration file containing the task settings, such as: `schtasks create tn "Housekeeping Task" /xml housekeepingtask.xml` As at other times when you are creating tasks, you can use the *S parameter to specify a remote computer,* U to specify the user context for creating the task, and *P to specify the user password. Although the XML configuration file for a task can define alternate credentials under which the task runs, you can also specify the alternate credentials using the* Ru and /Rp parameters.

Rather than having the actual user password in the file, you may want to set the password to blank or "*". If you do this, you can specify the required password when you create the task using the /Rp parameter.

The following is an XML configuration file that defines a scheduled task and its settings:

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2"
xmlns="http://schemas.microsoft.com/windows/2015/02/mit/task">
  <RegistrationInfo>
```

```
    <Date>2015-10-01T18:10:12</Date>
    <Author>WilliamS</Author>
  </RegistrationInfo>
  <Triggers>
    <EventTrigger>
      <StartBoundary>2015-10-01T18:10:00</StartBoundary>
      <Enabled>true</Enabled>
      <Subscription>&lt;QueryList&gt;&lt;Query&gt;&lt;Select
Path='system'&gt;*[System[(Level=1  or Level=2 or Level=3)]
]&lt;/Select&gt;&lt;/Query&gt;&lt;/QueryList&gt;</Subscription>
    </EventTrigger>
  </Triggers>
  <Settings>
    <IdleSettings>
      <Duration>PT10M</Duration>
      <WaitTimeout>PT1H</WaitTimeout>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>true</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>false</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT72H</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>wevtvwr.msc</Command>
    </Exec>
  </Actions>
  <Principals>
    <Principal id="Author">
      <UserId>IMAGINEDL\WILLIAMS</UserId>
      <LogonType>InteractiveToken</LogonType>
    </Principal>
  </Principals>
</Task>
```
As you can see, this file is fairly complex, but don’t worry. You don’t have to create the XML configuration file from scratch. Using the techniques discussed in the section titled “Copying Tasks to Other Computers” earlier in this chapter, you can export the settings for an

existing task into an XML configuration file and then use this file to create the same task on other computers.

At the command line, you can display a task's status and its XML configuration using Schtasks *Query*. *Simply use the* Tn parameter followed by the name of the task to work with and the /Xml parameter to display the XML configuration as well as the status. This example displays the status and XML configuration for the "Computer Authentication" task: `schtasks query tn "Computer Authentication" /xml` If you redirect this output to a file with the .xml extension and then edit this file to remove the status details, you'll have a complete XML configuration file that you can use to create the task. In the following example, the configuration for the "Computer Authentication" task is written to a file called ComputerAuthTask.xml: `schtasks query tn "Computer Authentication" /xml > ComputerAuthTask.xml` When working with remote computers, you can use the *S parameter to specify a remote computer,* U to specify the user context for creating the task, and /P to specify the user password.

Once you gain a working knowledge of how the XML configuration files work, you may want to start manually editing XML configuration files as necessary. When you do, make sure to test your changes on a test or development system rather than a production system. That said, let's go through the essentials.

The RegistrationInfo element specifies when the task was originally created and by whom:

```
<RegistrationInfo>
    <Date>2015-10-01T18:10:12</Date>
    <Author>WilliamS</Author>
  </RegistrationInfo>
```

The Triggers element specifies under which conditions the task runs. Within this element, the following is true:

- EventTrigger elements define tasks triggered by events.
- TimeTrigger elements define tasks triggered by one-time or periodic tasks.
- BootTrigger elements define tasks triggered at startup.
- IdleTrigger elements define tasks triggered when the computer

is idle.

- RegistrationTrigger elements define tasks triggered when the task is created or modified.

Boot, idle, and registration triggers are the easiest to define because they are either enabled or not enabled, as shown in this example:

```
<Triggers>
    <BootTrigger>
      <Enabled>true</Enabled>
    </BootTrigger>
  </Triggers>
```

Actions elements define the commands to run, the e-mail to send, or the message to display. Commands to run are defined within Exec elements. The following example runs a script called CleanUp.bat:

```
<Actions Context="Author">
    <Exec>
      <Command>c:\scripts\cleanup.bat</Command>
    </Exec>
  </Actions>
```

E-mail messages to send are defined within SendEmail elements. The following example defines an e-mail message to send to admins@imaginedlands.com via the mailer15.adatum.com mail server:

```
<Actions Context="Author">
    <SendEmail>
      <Server>mailer15.imaginedlands.com</Server>
      <Subject>Possible Database Outage</Subject>
      <To>admins@imaginedlands.com</To>
      <From>williams@imaginedlands.com</From>
      <Body>The CRM Database appears to be down.</Body>
      <HeaderFields />
    </SendEmail>
  </Actions>
```

Messages to display on the desktop are defined within ShowMessage elements. The following example displays a warning about a possible application outage:

```
<Actions Context="Author">
    <ShowMessage>
      <Title>Application Outage Warning</Title>
      <Body>The CRMComms application is having write errors.</Body>
    </ShowMessage>
  </Actions>
```

Finally, the Principals element defines the user context under which the task runs, including whether the task can run interactively, and the run level. In the following example, the task runs interactively with least privileges under the user account of WilliamS:

```
<Principals>
    <Principal id="Author">
      <UserId>IMAGINEDL\williams</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
```
You can configure the task to run whether the user is logged on or not, or to run with highest privileges. To run whether the user is logged on or not, you set LogonType to Password. To run with highest privileges, you set RunLevel to HighestAvailable. Here is an example using these options:
```
<Principals>
    <Principal id="Author">
      <UserId>IMAGINEDL\williams</UserId>
      <LogonType>Password</LogonType>
      <RunLevel>HighestAvailable</RunLevel>
    </Principal>
  </Principals>
```
If you don't want Task Scheduler to store the password associated with the user account, you can set the LogonType to S4U, as shown in this example:
```
<Principals>
    <Principal id="Author">
      <UserId>IMAGINEDL\williams</UserId>
      <LogonType>S4U</LogonType>
      <RunLevel>HighestAvailable</RunLevel>
    </Principal>
  </Principals>
```

Running Tasks Immediately with Schtasks /Run

You can run a task at any time using the following syntax: `schtasks run tn` *TaskName* where *TaskName* is the name of the task you want to run, such as `schtasks run tn "SysChecks"`

Running a task does not affect its schedule and does not change the next run time for the task. If the task can be successfully started, you should see a message stating this. Additionally, you can specify the name of the remote computer on which the task is configured and, as necessary, the account to run the task as, including an optional password, as in the following examples:
```
schtasks /run tn "SysChecks" s 192.168.1.100
schtasks /run tn "SysChecks" s 192.168.1.100 /u imaginedlands\wrstanek
```
NOTE If you specify a user and don't provide a password, you will be prompted immediately to enter the password.

Stopping Running Tasks with Schtasks /End

You can stop a task at any time using the following syntax: `schtasks end tn` *TaskName* where *TaskName* is the name of a task that is currently running and should be stopped, such as `schtasks end tn "SysChecks"`

The task is only stopped if it is running. If successful, the output message should be similar to the following: `SUCCESS: The scheduled task "SysChecks" has been terminated successfully.`

You can also specify the name of the remote computer on which the task is configured and, as necessary, the account with authority to stop the task, including an optional password, such as `schtasks /end tn "SysChecks" s 192.168.1.100`

or

```
schtasks /end tn "SysChecks" s 192.168.1.100 /u imaginedlands\wrstanek
```

Because a password isn't specified, Schtasks will prompt you for one.

Deleting Tasks with Schtasks /Delete

You can delete tasks by name on local and remote computers using the following syntax: `schtasks delete tn` *TaskName* `[/s` *Computer* `/u [`*Domain*`/]`*User* `[/p` *Password*`]]`

where *TaskName* is the name of a task that should be deleted and the rest of the parameters optionally identify the remote computer, the user account to use when deleting the task, and the password for the account, such as `schtasks delete tn "SysChecks"`

or

```
schtasks /delete tn "SysChecks" s 192.168.1.100 u imaginedlands\wrstanek
p frut5
```

> *NOTE* If you specify a user name and don't provide a password, you will be prompted immediately to enter the password.

After entering the Schtasks /Delete command, you should see a warning asking you to confirm that you want to remove the task. Press the appropriate letter on your keyboard. If you don’t want to see a warning prompt use the /F parameter, such as `schtasks /delete tn "SysChecks" f` Here, you force Schtasks to delete the task without a warning.

In addition, if you want to delete all scheduled tasks on the local computer or the specified remote computer, enter an asterisk (*) as the task name, such as `schtasks delete tn *`

Confirm the action when prompted.

Chapter 10. Configuring, Maintaining, and Troubleshooting TCP/IP Networking

Computers often experience issues related to Transmission Control Protocol/Internet Protocol (TCP/IP) networking and related configuration settings. As being able to configure, maintain, and troubleshoot TCP/IP networking is important for anyone who wants to master the command-line, this chapter starts with a discussion of the command-line tools available for performing these tasks and then delves into each area separately, giving you the knowledge and techniques you'll need to successfully manage and support TCP/IP networking on Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 systems.

Using the Network Services Shell

The network services shell (Netsh) is a command-line scripting utility that allows you to manage the configuration of various network services on local and remote computers. Netsh provides a separate command prompt that you can use in either interactive or noninteractive mode.

Working with Netsh Contexts

In interactive mode, you enter the shell by typing netsh and then specifying the context name of the network service you want to work with. The contexts and subcontexts available depend on the role services, roles, and features installed on the computer. Key context names and their meanings are as follows:

- **advfirewall** Advanced firewall. The context used to manage and monitor the Windows Firewall with Advanced Security. Windows Firewall with Advanced Security is an enhanced version of the standard Windows Firewall that allows you to define security policies and includes extensions for defining advanced packet filtering rules for both inbound and outbound connections.

- **bridge** Network Bridge. The context used to enable or disable transport layer (OSI model layer 3) compatibility mode for network bridges. Also used to view the configuration settings of network bridges.
- **dhcp** Dynamic Host Configuration Protocol (DHCP). The context used for viewing and managing DHCP servers. You use the DHCP context to assign
  TCP/IP configuration information dynamically to network clients. This context is only available on Windows Server 2012 and Windows Server 2012 R2 when the DHCP Server role service is installed. If the DHCP Server role service is not installed, the shortcut context name "dhcp" opens the dhcpclient context.
- **dhcpclient** DHCP client. The context used for enabling or disabling tracing of DHCP communications.
- **firewall** Windows Firewall. The context used to manage allowed programs, firewall ports, logging, notification and other aspects of the Windows Firewall.
- **http** Hypertext Transfer Protocol (HTTP). The context used to manage the configuration of HTTP listeners.
- **interface ipv4** Interface IP version 4 (IPv4). The context used to view and manage the IPv4 network configuration of a computer. Many Interface IPv4 Show commands are only available when working locally.
- **interface ipv6** Interface IP version 6 (IPv6). The context used to view and manage the IPv6 network configuration of a computer. Many Interface IPv6 Show commands are only available when working locally.
- **interface portproxy** Interface Port Proxy. The context used to manage proxies for IPv4 networks and IPv6 networks as well as between IPv4 and IPv6 networks.
- **ipsec** Internet Protocol Security (IPsec). The context used to view

and configure dynamic and static settings for IPsec.

- **lan** Wired Local Area Network (LAN). The context used to manage wired network profiles and to work with wired interfaces. Most Lan commands make use of the Wired Autoconfig service and you must start this service to use these commands.
- **nap client** Network Access Protection (NAP) client. The context used to manage the NAP client configuration.
- **nap hra** NAP Health Registration Authority (HRA). The context used to manage the NAP HRA configuration. This context is only available on Windows Server 2012 and Windows Server 2012 R2 when the Health Registration Authority role service is installed as part of the Network Policy And Access Services role.
- **netio** Network Input/Output (netio). The context allows you to add, delete, and list network binding filters.
- **ras** Remote access server (RAS). The context used to view and manage remote access server configurations.
- **ras aaaa** Authentication, authorization, accounting, and auditing (AAAA). The context used to view and work with the AAAA database. That database is used by the Internet Authentication Service (IAS) and the Routing And Remote Access service.
- **ras diagnostics** RAS diagnostics. The context used to configure diagnostics logging and traces for troubleshooting RAS.
- **routing** Routing. The context used to manage routing servers. Used with Routing And Remote Access server. This context is only available on Windows Server 2012 and Windows Server 2012 R2 when the Remote Access role is installed.
- **rpc** Remote procedure call (RPC) helper. The context used to view and manage IP address interface settings as well as IP subnet addresses that are configured on a computer. The commands in this context can only be used when you are working locally.

- **rpc filter** RPC firewall. The context used to create and manage RPC firewall filters and rules. The commands in this context can only be used when you are working locally.
- **winhttp** Windows HTTP (WinHTTP). The context used to manage WinHTTP proxy and tracing settings.
- **wins** Windows Internet Name Service (WINS). The context used to view and manage WINS server settings. You use WINS to resolve NetBIOS computer names to IPv4 addresses for pre–Windows 2000 computers. This context is only available on Windows Server 2012 and Windows Server 2012 R2 when the WINS Server feature is installed.
- **winsock** Winsock. The context used to view and manage Winsock communications settings.
- **wlan** Wireless LAN. The context used to view and manage wireless networking settings. On Windows Server 2008, commands in this context are only available when the Wireless LAN service is installed.

NOTE As noted, some contexts and commands are only available when you use Netsh on a local computer. The key one you'll notice is RPC, which is only available when you are working locally. In addition, some Netsh contexts and commands require the Routing And Remote Access service to be configured even when you are working with a local computer at the command line. If this is the case, you must set the Connections To Other Access Servers remote access policy to grant remote access permission, and then ensure that the remote access service is running.

The context name tells Netsh which helper DLL to load. The helper DLL provides context-specific commands that you can use. For example, if you typed netsh to work interactively with Netsh and then typed **rpc** you would enter the RPC context. You could then type **show interfaces** to see the IPv4 address interfaces configured on the computer. As a series of steps, this would look like this: 1. Type **netsh**. The command prompt changes to: **netsh>**.

2. Type **rpc**. The command prompt changes to: **netsh rpc>**.
3. Type **show interfaces**. The IPv4 address interfaces configured on the computer are displayed, such as

   ```
   Subnet       Interface    Status
   Description
   127.0.0.0    127.0.0.1    Enabled    Software Loopback Interface 1
   192.168.1.0 192.168.1.101 Enabled   Intel(R) PRO/1000 PM Network
   Connection
   ```

   Each context has a different set of commands available and some of these commands lead to subcontexts that have their own commands as well. Keep in mind the related service for the context must be configured on the computer or in the domain to allow you to do meaningful work within a particular context. Regardless of what context you are working with, you can view the list of available commands by typing **help** or **?**. Similarly, regardless of what context you are in, typing **exit** or **quit** will exit the network services shell, returning you to the Windows command prompt.

To switch to a context regardless of where you are within the network services shell, type the full context name. For example, if you are working with the Interface Ipv6 context and want to switch to the Ras Diagnostics context, simply enter **ras diagnostics**. Finally, regardless of which context you are working with, you can always use the .. command to go up one context level. This means that if you are working with the Netsh Rpc context and later want to switch back to the top-level netsh context, you would type .. to go up one context level.

Well, that's how Netsh works interactively: It's slow and plodding, but it's good for beginners or for digging around to find out what commands are available. Once you grow accustomed to working with Netsh, you'll want to use this utility in noninteractive mode. Noninteractive mode allows you to type in complete command sequences at the command-line prompt or within batch scripts. For example, the previous procedure, which took three steps, can be performed with one command line:

```
netsh rpc show
interfaces
```

Whether you insert this line into a script or type it directly at a command-line prompt, the resulting output is the same: a list of interfaces on the computer you are working with. As you can see, typing commands directly is a lot faster.

Working with Remote Computers

You can use Netsh to work with remote computers. To work interactively with a remote computer, you start Netsh with the –R parameter and then specify the IP address or domain name of the computer to which you want to connect, such as `netsh -r 192.168.10.15`

or

```
netsh -r corpsvr02
```

While you work with the remote computer, Netsh will include the computer IP address or name in its command prompt, such as `[corpsvr02] netsh>`

Here you use Netsh to work remotely with CorpSvr02. You can provide any necessary user credentials with the following parameters:

- **-u:** Specifies a different user to log on to a remote computer in the form Domain\User or User. Only applicable when working with a remote computer.
- **-p:** To attempt to verify connectivity to various remote hosts and the TCP/IP configuration, you can do one of the following Sets the password for the specified user. If you don’t specify a password, or if you use "*", you are prompted to enter a password. Only applicable when a different log-on user is specified when working with a remote computer.

In the following example, you work with Netsh on FileServer25 and use the IMAGINEDL\WilliamS account to log on: `netsh -r fileserver25 -u imaginedlands\williams -p *`

If you want to work noninteractively with remote computers, you must use the following syntax: `netsh -c Context -r RemoteComputer Command` where *Context* is the identifier for the context you want to work with, *RemoteComputer* is the name or IP address of the remote computer, and *Command* is the command to execute. Consider the following example: `netsh -c "interface ipv4" -r corpsvr02 show interfaces` In this example, you use the Interface IPv4 context to obtain a list of interfaces configured on CorpSvr02. Here, you cannot use the RPC context to perform this task, because this context is only available on a local computer.

> *REAL WORLD* To use Netsh to interact with a remote computer, the Routing And Remote Access service must be configured on the network. Specifically, you must set the Connections To Other Access Servers remote access policy to grant remote access permission. Then, ensure that the remote access service is running.

Working with Script Files

As discussed previously, you can type in complete Netsh command sequences at the command line or within batch scripts. The catch is that you must know the complete command line you want to use and cannot rely on Netsh for help. Some command lines can be very long and complex. For example, the following commands connect to a DHCP server, configure a DHCP scope, and then activate the scope: `netsh dhcp server \\corpsvr02 add scope 192.168.1.0 255.255.255.0 MainScope PrimaryScope`

```
netsh dhcp server \\corpsvr02 scope 192.168.1.0 add iprange 192.168.1.1
192.168.1.254

netsh dhcp server \\corpsvr02 scope 192.168.1.0 add excluderange
192.168.
1.1 192.168.1.25

netsh dhcp server \\corpsvr02 scope 192.168.1.0 set state 1
```

If you save these commands to a batch script, you can run the script just as you would any other batch script. For example, if you named the script dhcpconfig.bat, you would type **dhcpconfig** to run the script.

When working with remote computers, you can place the script on a network share accessible from the remote computer and then log on remotely to execute the script. Or you can copy the script directly to the remote computer and then log on to execute it remotely. Either way works, but both involve a couple of extra steps. Fortunately, there's a faster way to run a script on a remote computer. To do this, you must change the script a bit and use the following syntax: `netsh -c` *`Context`* `-r` *`RemoteComputer`* `-f Script` where *Context* is the identifier for the context you want to work with, *RemoteComputer* is the name or IP address of the remote computer, and Script is the file or network path to the script to execute. Consider the following example: `netsh -c "dhcp server" -r corpsvr02 -f dhcpconfig.bat` In this example, you run a Netsh script called dhcpconfig.bat on CorpSvr02 using the DHCP Server context. Note that Server is a subcontext of the DHCP context. The script contains the following commands:

```
add scope 192.168.1.0 255.255.255.0 MainScope
PrimaryScope
scope 192.168.1.0 add iprange 192.168.1.1 192.168.1.254
scope 192.168.1.0 add excluderange 192.168.1.1 192.168.1.25
scope 192.168.1.0 set state 1
```

These commands create, configure, and then activate a DHCP scope on the designated DHCP Server, CorpSvr02. Because you are already using the DHCP Server context on CorpSvr02, you don't need to type netsh dhcp server \\corpsvr02 at the beginning of each command.

Managing TCP/IP Settings

Computers use IP addresses to communicate over TCP/IP. Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 have a dual IP layer architecture in which both Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) are implemented and share common Transport and Frame layers. IPv4 and IPv6 are used in very different ways. IPv4 has 32-bit addresses and is the primary version of IP used on most networks, including the Internet. IPv6 has 128-bit addresses and is the next generation version of IP.

When networking hardware is detected during installation of the operating system, both IPv4 and IPv6 are enabled by default and you don't need to install a separate component to enable support for IPv6. IP addressing can be configured manually or dynamically at the command line. With a manual configuration, you assign the computer a static IP address. Static IP addresses are fixed and don't change unless you change them. With a dynamic configuration, you configure the computer to get its IP address assignment from a DHCP server on the network. This IP address is assigned when the computer starts and might change over time. In domains, Windows servers typically use static IP addresses and Windows workstations typically use dynamic IP addresses.

Configuring IPv4

IPv4's 32-bit addresses commonly are expressed as four separate decimal values, such as 127.0.0.1 or 192.168.10.50. The four decimal values are referred to as octets because each represents 8 bits of the 32-bit number. (Using 8 bits for each decimal number limits the range of possible values to zero through 255.) With standard unicast IPv4 addresses, a variable part of the IP address represents the network ID and a variable part of the IP address represents the host ID. There is no correlation between a host's IPv4 address and the internal machine (MAC) address used by the host's network adapter.

Setting a Static IPv4 Address

When you set a static IPv4 address, you tell the computer the IPv4 address to use, the subnet mask for this IPv4 address, and, if necessary, the default gateway to use for internetwork communications. After you configure these IPv4 settings, you will also need to configure name-resolution settings for Domain Name System (DNS) and possibly WINS.

You assign a static IPv4 address using the Netsh Interface IPv4 context. The command is SET ADDRESS and its syntax is `set address`
`[name=]`*InterfaceName* `source=static address=`*IPAddress*
`mask=`*SubnetMask* `[gateway={none |` *DefaultGateway*
`[[gwmetric=]`*GatewayMetric*`]}`

NOTE If the computer already had an IPv4 address configuration on the specified interface, using SET ADDRESS replaces the existing values. To add to the existing settings instead of replacing them, use the ADD ADDRESS command.

You can check to see the available interfaces and their current configuration by typing **netsh interface ipv4 show addresses** at the command prompt or, if you are in the Netsh Interface IPv4 context, by typing **show addresses**. As shown in the following example, the output specifies the name of the interfaces available and their current configuration:

```
Configuration for interface "Local Area Connection"
    DHCP enabled:                         Yes
    IP Address:                           192.168.1.101
    Subnet Prefix:                          192.168.1.0/24 (mask
255.255.255.0)
    Default Gateway:                      192.168.1.1
    Gateway Metric:                       0
    InterfaceMetric:                      10

Configuration for interface "Loopback Pseudo-Interface 1"
    DHCP enabled:                         No
    IP Address:                           127.0.0.1
    Subnet Prefix:                        127.0.0.0/8 (mask 255.0.0.0)
    InterfaceMetric:                      50
```

In most cases, the interface name you want to work with is “Local Area Connection.” The pseudo interface listed in this example is used for local loopback communications. The IPv4 address you assign to the computer must not be used anywhere else on the network. The subnet mask field ensures that the computer communicates over the network properly. If the network uses subnets, the value you use may be different on each network segment within the company. If the computer needs to access other TCP/IP networks, the Internet, or other subnets, you must specify a default gateway. Use the IPv4 address of the network’s default router.

The gateway metric indicates the relative cost of using a gateway. If multiple default routes are available for a particular IPv4 address, the gateway with the lowest cost is used first. If the computer can’t communicate with the initial gateway, Windows tries to use the gateway with the next lowest metric. Unlike the GUI, Windows doesn’t automatically assign a metric to the gateway. You must assign the metric manually.

Consider the following example:

```
set address name="Local Area Connection" source=static
address=192.168.1.50 mask=255.255.255.0 gateway=192.168.1.1
gwmetric=1
```

Here you specify that you are working with the “Local Area Connection” interface, setting a static IPv4 address of 192.168.1.50 with a network mask of 255.255.255.0. The default gateway is 192.168.1.1 and the gateway metric is 1.

> *TIP* You can confirm the settings you just made by typing **netsh interface ipv4 show addresses** at the command prompt or, if you are in the Netsh Interface IPv4 context, by typing **show addresses**. Because many Interface IPv4 and Interface IPv6 Show commands are only available when working locally, including Show Addresses, you must be logged on locally to use this command.

Setting a Dynamic IPv4 Address

You can assign a dynamic IPv4 address to any of the network adapters on a computer, provided that a DHCP server is available on the network. Then you rely on the DHCP server to supply the necessary IPv4 addressing information. Because the dynamic IPv4 address can change, you normally don’t use a dynamic IPv4 address for servers running Windows Server 2012 or Windows Server 2012 R2.

You assign a dynamic IPv4 address using the Netsh Interface IPv4 context. The command is SET ADDRESS and its syntax is `set address name=`*InterfaceName* `source=dhcp` Consider the following example:

`set address name="Local Area Connection" source=dhcp` Here you are working in the Netsh Interface IPv4 context and specify that you want to set a dynamic IPv4 address for the “Local Area Connection” interface.

Adding IPv4 Addresses and Gateways

Windows 8.1, Windows Server 2012, and Windows Server 2012 R2 systems can have multiple IPv4 addresses, even if the computer only has a single network adapter. Multiple IPv4 addresses are useful if you want a single computer to appear as several computers or your network is divided into subnets and the computer needs access to these subnets to route information or provide other internetworking services.

> *NOTE* Keep in mind that when you use a single network adapter, IPv4 addresses must be assigned to the same network segment or segments that are part of a single logical network. If your network consists of multiple physical networks, you must use multiple network adapters, with each network adapter being assigned an IPv4 address in a different physical network segment.

You assign multiple IPv4 addresses and gateways to a single network adapter using the ADD ADDRESS command of the Netsh Interface IPv4 context. The syntax for this command is similar to that of SET ADDRESS. It is `add address [name=]`*InterfaceName* `address=`*IPAddress* `mask=`*SubnetMask* `[[gateway=]`*DefaultGateway* `[gwmetric=]`*GatewayMetric*`]`

Consider the following example:

```
add address name="Local Area Connection" address=192.168.2.12
mask=255.255.255.0 gateway=192.168.2.1 gwmetric=1
```

> *NOTE* If you specify a gateway, you must also specify the gateway metric. As before, you can confirm the settings you just made by typing **show addresses**.

Here you specify that you are working with the “Local Area Connection” interface and adding the IPv4 address of 192.168.2.12 with a network mask of 255.255.255.0. The default gateway for this IPv4 address is 192.168.2.1 and the gateway metric is 1.

Setting DNS Servers to Use for IPv4

Computers use DNS to determine a computer's IP address from its host name or its host name from an IP address. For computers using static IPv4 addresses, you must tell them which DNS servers to use; you can do this using the Netsh Interface IPv4 context. The syntax for setting a specific DNS server to use is `set dnsserver name=`*`InterfaceName`*` source=static address=`*`DNSAddress`* Consider the following example:

```
set dnsserver name="Local Area Connection" source=static
address=192.168.1.56
```

Here you specify that you are working with the "Local Area Connection" interface and specifying the DNS server address as 192.168.1.56.

When you are configuring a static DNS server address, you can use the optional register parameter to control DNS registration. Keep the following in mind:

- By default, all IP addresses for interfaces are registered in DNS under the computer's fully qualified domain name. This automatic registration uses the DNS dynamic update protocol. If you want to disable this behavior, use **register=none**.
- By default, the computer's full name is registered only in its primary domain as set by the default value **register=primary**. When using dynamic DNS, you can also specify that the connection-specific DNS name should be registered with DNS. To do this, use **register=both**. This allows for occasions when the computer has multiple network adapters that connect to multiple domains.

If a computer is using DHCPv4 and you want DHCPv4 to provide the DNS server address, you can provide the DNS server address as well or specify that the IPv4 address should be obtained from DHCPv4. You tell the computer to get the DNS server settings from DHCPv4 by typing `set dnsserver name=`*`InterfaceName`*` source=dhcp` Consider the following example:

`set dnsserver name="Local Area Connection" source=dhcp` Here you specify that the “Local Area Connection” interface should get its DNS server address settings from DHCPv4.

> *NOTE* If the computer already had DNS server IPv4 addresses set, using SET
> DNSSERVER replaces the existing values. To add DNS server IPv4 addresses instead of replacing them, use the ADD DNSSERVER command. You can confirm the DNS server settings by typing **show dnsservers**.

Specifying Additional DNS Servers to Use

Most networks have multiple DNS servers that are used for resolving domain names. This allows for name resolution if one DNS server isn’t available. When you use DHCPv4 to specify the DNS servers, it can automatically tell computers about other DNS servers that may be available. This isn’t the case when you manually specify DNS servers to be used.

To tell a computer about other DNS servers that may be available in addition to the primary DNS server specified previously, you can use the Netsh Interface IPv4 context and the ADD DNSSERVER command. The syntax is `add dnsserver name=`*InterfaceName* `address=`*DNSAddress* Consider the following example:

```
add dnsserver name="Local Area Connection" address=192.168.1.75
```

Here you specify that you are working with the “Local Area Connection” interface and designating an alternate DNS server with an IP address of 192.168.1.75.

By default, a DNS server is added to the end of the DNS server’s list in the TCP/IP configuration. If you want the DNS server to be in a specific position in the list use the Index= parameter. For example, if you wanted an additional server to be listed first (making it the primary) you’d set an index of 1, such as: `add dnsserver name="Local Area Connection" address=192.168.1.75 index=1`

Deleting IPv4 Address Resolution Protocol Cache

When computers look up domain name information for IPv4 addresses, the related information is stored in the Address Resolution Protocol (ARP) cache so that the next time the information is needed no name lookup is necessary. The address resolution information expires according to a time-to-live (TTL) value set when the information was received, after which time it must again be looked up to get current information and a new TTL. In general, this automated system of obtaining, clearing out, and renewing name information works well. Sometimes, however, old name-resolution information on a system will cause problems before the information is purged. For example, if a computer changes its name information and the TTL hasn't expired on a previous lookup, temporarily you won't be able to find the computer.

DNS administrators have several tricks they can use to reduce the impact of name changes, such as setting an increasingly shorter TTL just prior to a name change to ensure that old information is deleted more quickly and doesn't cause a problem. However, you may find that it's easier just to get rid of the old information and force a computer to make new DNS lookups. You can do this by typing **netsh interface ipv4 delete arpcache** at the command prompt or, if you are in the Netsh Interface IPv4 context, by typing **delete arpcache**. This deletes name information for all interfaces configured on the computer you are working with. When there are multiple interfaces and you only want name-resolution information purged for one interface, you can name the interface to work with by including **name=InterfaceName**, such as `delete arpcache name="Local Area Connection"`

Deleting TCP/IPv4 Settings

Using the Netsh Interface IPv4 context, you can delete TCP/IPv4 configuration settings as well. Table 17-1 summarizes the available commands according to the task to be performed.

TABLE 17-1 Netsh Interface IPv4 Commands for Deleting TCP/IPv4 Settings **TASK**

SYNTAX
EXAMPLE

Delete a designated IPv4 address from the named interface.

delete address
name=*InterfaceName*

address=*IPAddress* delete address name= “Local Area
Network” address=192.168.1.5 6

Delete a static gateway IPv4 address from the named interface.

delete address

name=*InterfaceName* gateway=*GatewayAddress* delete address name= “Local Area
Network” gateway=192.168.1.1

Delete all static gateway IPv4 addresses from the named interface.

delete address

name=*InterfaceName* gateway=all delete address name= “Local Area

Network” gateway=all Delete a DNS server from the named interface.

delete dnsserver

name=*InterfaceName* address=*IPAddress* delete dnsserver name=“Local Area Network” address=
192.168.1.5 6

Delete all DNS servers from the named interface.

delete dnsserver

name=*InterfaceName*

address=all delete dnsserver name=“Local Area Network” address=all Delete a WINS server from the named interface.

delete winsserver

name=*InterfaceName*

address=*IPAddress* Delete winsserver name=
“Local Area Network”
address=192.168.1.5 6

Delete all WINS servers from the named interface.

delete winsserver
name=*InterfaceName*

address=all delete winsserver name=“Local Area Network” address=all

Configuring IPv6

IPv6’s 128-bit addresses are divided into eight 16-bit blocks delimited by colons. Each 16-bit block is expressed in hexadecimal form. With standard unicast IPv6 addresses, the first 64 bits represent the network ID and the last 64 bits represent the network interface. An example of an IPv6 address is FEC0:0:0:02BC:FF:BECB:FE4F:961D. Because many IPv6 address blocks are set to 0, a contiguous set of 0 blocks can be expressed as "::", a notation referred to as the double-colon notation. Using double-colon notation, the two 0 blocks in the previous address are compressed as FEC0::02BC:FF:BECB:FE4F:961D. Three or more 0 blocks would be compressed in the same way. For example, FFE8:0:0:0:0:0:0:1 becomes FFE8::1.

Setting IPv6 Addresses

By default, computers have their IPv6 configuration automatically assigned. When a computer using IPv6 connects to the network, it sends a link-local multicast request to retrieve configuration settings. Computers using IPv6 can also use DHCPv6 to obtain IPv6 configuration information from a DHCPv6 server. When you configure DHCPv6 servers on a network, you specify how DHCPv6 works for clients and do not need to modify dynamic client configurations to support this. Automatically assigned IPv6 addresses typically are link-local addresses that are accessible only on the local network.

For computers that require routable IPv6 addresses, you'll need to assign a static IPv6 address. When you set a static IPv6 address, you tell the computer the IPv6 address to use and specify whether the address is unicast or anycast. Unicast addresses are the default, and you'll assign two main variations of unicast addresses: unique local unicast and global unicast. Unique local IPv6 unicast addresses are routable on your internal network but not accessible from the Internet. Global unicast addresses, on the other hand, can be routed to the Internet, such as may be required for external servers. Anycast addresses are addresses that can be assigned to multiple interfaces, such as a single IPv6 address for all the interfaces on a computer.

You assign a static IPv6 address using the Netsh Interface IPv6 context. The command is SET ADDRESS and its syntax is `set address [interface=]`*InterfaceName* `address=`*IPAddress* `type=`*AddressType* *NOTE* If the computer already had an IPv6 address configuration on the specified interface, using SET ADDRESS replaces the existing values. To add to the existing settings instead of replacing them, use the ADD ADDRESS command.

You can check to see the available interfaces and their current configuration by typing **netsh interface ipv6 show addresses** at the command prompt or, if you are in the Netsh Interface IPv6 context, by typing **show addresses**. As shown in the following example, the output specifies the name of the interfaces available and their current configuration:

```
Interface 1: Loopback Pseudo-Interface 1

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- -------------------
Other      Preferred     infinite   infinite ::1

Interface 7: Local Area Connection

Addr Type  DAD State   Valid Life Pref. Life Address
---------  ----------- ---------- ---------- -------------------
Other   Preferred  infinite infinite fe80::6712:1345:cc87:3820%7

Interface 11: Local Area Connection* 8

Addr Type  DAD State Valid Life Pref. Life Address
---------  --------- ---------- ---------- ------------
Other   Preferred  infinite infinite fe80::5efe:192.168.1.101%11
```

> *NOTE* Computers running IPv6 use the fe80::/64 link-local network address for network adapters connected to networks without IPv6 routers or DHCPv6 servers. When a computer has multiple network adapters configured, it adds a numeric identifier following a percent sign to the IP address in the output. In the example, %7 is added to the output for Interface 7 and %11 is added to the output for Interface 11.

Consider the following example:

```
set address interface="Local Area Connection"
address= 2001:1cb7::2b58:02bb:00ff:fe45:bc7d type= unicast
```

Here you specify that you are working with the “Local Area Connection” interface,

setting a static unicast IPv6 address of
2001:1cb7::2b58:02bb:00ff:fe45:bc7d.

> *TIP* You can confirm the settings you just made by typing **netsh interface ipv6 show addresses** at the command prompt or, if you are in the Netsh Interface IPv6 context, by typing **show addresses**. Because many Interface IPv6 and Interface IPv6 show commands are only available when working locally, including show addresses, you must be logged on locally to use this command.

You can assign additional IPv6 addresses to a network adapter using the ADD ADDRESS command of the Netsh Interface IPv6 context. The syntax for this command is similar to that of SET ADDRESS. It is `add address [interface=]`*InterfaceName* `address=IPAddress type=`*AddressType*

Setting DNS Servers to Use for IPv6

For computers using static IPv6 addresses, you must tell them which DNS servers to use; you can do this using the Netsh Interface IPv6 context. The syntax for setting a specific DNS server to use is `set dnsserver name=`*InterfaceName* `source=static address=`*DNSAddress register*=

Consider the following example:

```
set dnsserver name="Local Area Connection" source=static
address=fec0:0:0:ffff::1
```

Here you specify that you are working with the “Local Area Connection” interface and specifying the DNS server address as fec0:0:0:ffff::1.

When you are configuring a static DNS server address, you can use the optional register parameter to control DNS registration. Use **register=none** to disable dynamic DNS updates. Use **register=primary** to register the computer’s full name in its primary domain (as per the default setting). Use **register=both** to specify that the connection-specific DNS name should be registered with DNS as well as the primary DNS suffix.

If a computer is using DHCPv6 and you want DHCPv6 to provide the DNS server address, you can provide the DNS server address as well or specify that the IPv6 address should be obtained from DHCPv6. You tell the computer to get the DNS server settings from DHCPv6 by typing `set dnsserver name=InterfaceName source=dhcp` Consider the following example:

`set dnsserver name="Local Area Connection" source=dhcp` Here you specify that the "Local Area Connection" interface should get its DNS server address settings from DHCP.

> *NOTE* If the computer already had DNS server IPv6 addresses set, using SET DNSSERVER replaces the existing values. To add DNS server IPv6 addresses instead of replacing them, use the ADD DNSSERVER command. You can confirm the DNS server settings by typing **show dnsservers**.

To tell a computer about other DNS servers that may be available in addition to the primary DNS server specified previously, you can use the Netsh Interface IPv6 context and the ADD DNSSERVER command. The syntax is `add dnsserver name=`*InterfaceName* `address=`*DNSAddress* Consider the following example:

```
add dnsserver name="Local Area Connection" address=fec0:0:0:ffff::2
```

Here you specify that you are working with the "Local Area Connection" interface and designating an alternate DNS server with an IP address of fec0:0:0:ffff::2.

By default, a DNS server is added to the end of the DNS server's list in the TCP/IP configuration. If you want the DNS server to be in a specific position in the list, use the Index= parameter. For example, if you wanted an additional server to be listed first (making it the primary) you'd set an index of 1, such as: `add dnsserver name="Local Area Connection" address=`
```
fec0:0:0:ffff::2
index=1
```

Deleting TCP/IPv6 Settings

Using the Netsh Interface IPv6 context, you can delete TCP/IPv6 configuration settings as well. Table 17-2 summarizes the available commands according to the task to be performed.

TABLE 17-2 Netsh Interface IPv6 Commands for Deleting TCP/IPv6 Settings

TASK	SYNTAX	EXAMPLE
Delete a designated IPv6 address from the named interface.	delete address name=*InterfaceName* address=*IPAddress*	delete address name=“Local Area Network” address= 2001:1cb7::2b58:02bb:00ff:fe45:bc7d
Delete a DNS server from the named interface.	delete dns name=*InterfaceName* address=*IPAddress*	delete dns name=“Local Area Network” address= fec0:0:0:ffff::2
Delete all DNS servers from the named interface.	delete dns name=*InterfaceName* address=all	delete dns name=“Local Area Network” address=all

Supporting TCP/IP Networking

The Netsh shell provides two contexts for working with TCP/IP. You use the Interface IPv4 context to view TCP/IPv4 statistics and to change settings. You use the Interface IPv6 context to view TCP/IPv6 statistics and to change settings. The use of these contexts assumes that the necessary TCP/IP network components are already installed on the computer you are working with. If TCP/IP network components aren’t installed, you’ll need to install them.

Obtaining and Saving the TCP/IP Configuration

If you’ve worked with Windows for a while, you probably know that you can type ipconfig at a command prompt to get basic configuration information for IPv4 and IPv6, such as

```
Windows IP Configuration
Ethernet adapter Local Area Connection:
   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::6712:1345:cc87:3820%7
   IPv4 Address. . . . . . . . . . . : 192.168.1.101
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1

Tunnel adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter Local Area Connection* 8:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5efe:192.168.1.101%11
   Default Gateway . . . . . . . . . :
```

As you can see, this information shows the IPv6 link local address as well as the IPv4 address, subnet mask, and default gateway being used for the Local Area Connection Ethernet adapter. When you want more details, you type **ipconfig /all** to display additional information including the physical (MAC) address of the adapter, DHCP status, DNS servers used, and host information, such as

```
Windows IP Configuration
    Host Name:    salespc09
    Primary Dns Suffix:    imaginedlands.com
    Node Type:    Hybrid
    IP Routing Enabled:    No
    WINS Proxy Enabled:    No
Ethernet adapter Local Area Connection:
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) PRO/1000 PM Network
Connection
   Physical Address. . . . . . . . . : EA-BF-C2-D4-EF-12
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . :
fe80::6712:1345:cc87:3820%7(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.35(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . : Friday, April 04, 2015 9:37:43 AM
   Lease Expires . . . . : Saturday, April 05, 2015 12:05:32 PM
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCP Server . . . . . . . . . . . : 192.168.1.50
   DHCPv6 IAID . . . . . . . . . . . : 128384737
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Here a computer with the

fully qualified DNS name salespc09.imaginedlands.com is configured to use DHCP and has an IP address of 192.168.1.35 with subnet mask 255.255.255.0. Because the IP address was dynamically assigned, it has specific Lease Obtained and Lease Expiration date and time stamps.

If you type **netsh interface ipv4 show config** at a command prompt, you can obtain similar, albeit abbreviated, configuration information for IPv4, such as

```
Configuration for interface "Local Area Connection"
    DHCP enabled:    No
    IP Address:    192.168.1.50
    Subnet Prefix:    192.168.1.0/24 (mask 255.255.255.0)
    Default Gateway:    192.168.1.50
    GatewayMetric:    256
    InterfaceMetric:    20
    Statically Configured DNS Servers:    127.0.0.1
    Register with which suffix:    Primary only     Statically
Configured WINS Servers:    None
```

You can examine basic IPv6 configuration information by typing **netsh interface ipv6 show addresses** and **netsh interface ipv6 show dnsservers**. As you can see, these are all ways to obtain similar information about the TCP/IP configuration.

Netsh also gives you the means to save the IPv4 and IPv6 configuration so that the settings can be recreated simply by running a Netsh script. If you want to save the IPv4 settings to a file, type: `netsh interface ipv4 dump >` *FileName* where FileName is the name of the file to which you want to write the IPv4 configuration information. Listing 17-1 shows an example of an IPv4 configuration file.

LISTING 17-1 IPv4 Configuration Script

```
# ----------------------------------
# IPv4 Configuration
# ----------------------------------
pushd interface ipv4


set address name="Local Area Connection" source=static
address=192.168.1.50 mask=255.255.255.0
set address name="Local Area Connection" gateway=192.168.1.1
gwmetric=1
set dnsserver name="Local Area Connection" source=static
address=192.168.1.56 register=primary
```

```
set winsserver name="Local Area Connection" source=static address=none

popd
# End of interface IPv4 configuration
```

Listing 17-1 is a Netsh script that you can run using the following syntax: `netsh -c "interface ipv4" -f` *FileName* Consider the following example:

```
netsh -c "interface ipv4" -f corpsvr02-ipconfig.txt
```

In this example, you run a Netsh script called corpsvr02-ipconfig.txt using the Interface IPv4 context to apply the IPv4 configuration defined in the script. One of the key reasons for creating a configuration dump is so that you have a backup of the IP configuration. If the configuration is altered incorrectly in the future, you can restore the original configuration from the script.

Examining IP Address and Interface Configurations

The Netsh Interface IPv4 and Netsh Interface IPv6 contexts provide several commands for viewing IP address and interface configurations. Here, an interface refers to a network adapter used by a computer to communicate over TCP/IP. Most computers have two interfaces: a local loopback interface and a Local Area Connection interface.

For IPv4, the local loopback interface is a pseudo-interface that uses the IPv4 address 127.0.0.1 and a network mask of 255.0.0.0. All IPv4 messages sent over this interface are looped back to the computer and are not sent out over the network.

For IPv6, the local loopback interface uses the IPv6 address ::1. All IPv6 messages sent over this interface are looped back to the computer and are not sent out over the network.

The Local Area Connection interface is created automatically when you install TCP/IP networking. Each network adapter will have one such interface. By default the first interface is named Local Area Connection, the second is Local Area Connection 2, and so on.

At the Windows command line, you can view global configuration information for IPv4 by typing **netsh interface ipv4 show global**. The output should be similar to the following:

```
General Global Parameters
---------------------------------------------
Default Hop Limit                   : 128 hops
Neighbor Cache Limit                : 256 entries per interface
Route Cache Limit                   : 128 entries per compartment
Reassembly Limit                    : 27229728 bytes
ICMP Redirects                      : enabled
Source Routing Behavior             : dontforward
Task Offload                        : enabled
Dhcp Media Sense                    : enabled
Media Sense Logging                 : enabled
MLD Level                           : all
MLD Version                         : version3
Multicast Forwarding                : disabled
Group Forwarded Fragments           : disabled
Randomize Identifiers               : enabled
Address Mask Reply                  : disabled

Current Global Statistics
---------------------------------------------
Number of Compartments              : 1
Number of NL clients                : 7
Number of FL providers              : 4
```

This information lists the global status of IPv4 for all interfaces configured on the computer. The Default Hop Limit lists the default maximum hops that are allowed for packets routed over the network. The reassembly limit indicates the maximum reassembly size of IP datagrams. Here, this means IP datagrams sent or received using this interface can have a maximum size of 27,229,728 bytes. Blocks of data aren’t usually sent in this size, however. Instead, they are separated into fragments, which are reassembled upon receipt into a complete IP datagram. We’ll discuss IP datagram fragmentation in more detail in a moment.

Using **netsh interface ipv4 show interfaces**, you can view summary information regarding a computer’s network interfaces. Consider the following example output:

```
Idx  Met   MTU          State        Name
---  ---   -----        ---------    -------------------
  1   50   4294967295   connected    Loopback Pseudo-Interface 1
  7   10   1500         connected    Local Area Connection
```

Here, the computer has two network interfaces. Loopback Pseudo-Interface 1 is connected, has an interface index of 1, has an interface metric of 50, and has an Ethernet maximum transmission unit (MTU) of 4,294,967,295 bytes. Local Area Connection is connected, has an interface index of 7, has an interface metric of 10, and has an MTU of 1,500 bytes.

When using Ethernet II encapsulation, an MTU of 1500 means each block of data transmitted is 1,500 bytes in length, with 20 bytes of this data block used for the IP header. The remaining 1,480 bytes are used as the IP payload for the data block. Thus, an IP datagram of 65,535 bytes would need to be fragmented into many smaller data blocks for transmission. These fragments would then be reassembled on the destination node.

The status of the interface usually reads as Connected when both ends of the computer's network cable are plugged in and Disconnected when either or both ends of the network cable are unplugged.

Working with TCP Internet Control and Error Messages

Each packet sent over IP is a datagram, meaning that it is an unacknowledged and nonsequenced message forwarded by routers to a destination IP address. Each router receiving a datagram decides how best it should be forwarded. This means different datagrams can take different routes between the sending IP address (the source node) and the destination IP address (the destination node). It also means the return route for individual datagrams can be different as well.

Although IP provides end-to-end delivery capabilities for IP datagrams, it does not provide any facilities for reporting routing or delivery errors encountered. Errors and control messages are tracked by the Internet Control Message Protocol (ICMP). You can view ICMP statistics by typing **netsh interface ipv4 show icmp**. The output from this command looks like this:

```
MIB-II ICMP Statistics
------------------------------------------------------
INPUT
Messages:    20302
Errors:    120
Destination Unreachable:    45
Time Exceeded:    88
Parameter Problems:    0
Source Quench:    4
Redirects:    6
```

```
Echo Requests:    966
Echo Replies:    966
Time Stamp Requests:    0
Time Stamp Replies:    0
Address Mask Requests:    0
Address Mask Replies:    0

OUTPUT
Messages:    20302
Errors:    120
Destination Unreachable:    45
Time Exceeded:    88
Parameter Problems:    0
Source Quench:    4
Redirects:    6
Echo Requests:    966
Echo Replies:    966
Time Stamp Requests:    0
Time Stamp Replies:    0
Address Mask Requests:    0
Address Mask Replies:    0
```

> *TIP* For Interface IPv4 and IPv6 SHOW commands that provide summary statistics, you can set the Rr= parameter to the number of seconds to use as a refresh interval. For example, if you wanted the interface statistics to be refreshed automatically every 30 seconds you would type **netsh interface ipv4 show icmp rr=30**. Once you set a refresh rate, you press Ctrl+C to exit the command so that no more updates are made.

In the preceding example, you see detailed statistics on IP datagram messages being received (input messages) and those being sent (output messages). Decoding these statistics is easy if you know what you are looking for. The most basic type of IP datagram message is Echo. It is used to send a simple message to an IP node and have an Echo Reply echoed back to the sender. Many TCP/IP network commands use Echo and Echo Reply to provide information about the reachability and the path taken to a destination IP node.

Any error that occurs during transfer of an IP datagram, in or out, is recorded as an error. IP attempts a best-effort delivery of datagrams to their destination IP nodes. If a routing or delivery error occurs along the transmission path or at the destination, a router or the destination node discards the problem datagram and tries to report the error by sending a "Destination Unreachable" message.

A time-to-live (TTL) value is set in an IP datagram before it is sent. This value represents the maximum number of hops to use between the source and the destination nodes. "Time exceeded" messages are sent back to the IP datagram originator when the TTL value of the datagram expires. Typically this means there are more links than expected between the source and the destination node. Here, you'd need to increase the TTL value to successfully send traffic between the source and destination node. An expired TTL could also be an indicator of a routing loop in the network. Routing loops occur when routers have incorrect routing information and forward IP datagrams in such a way that they never reach their destinations.

A router or destination node sends a "Parameter Problem" message when an error occurs while processing in the IP header within an IP datagram. The IP header error causes the IP datagram to be discarded, and if no other ICMP messages can be used to describe the error that occurred, the "Parameter Problem" message is sent back to the source node. Typically this indicates an incorrect formatting of the IP header or incorrect arguments in IP option fields.

When a router becomes congested—whether because of a sudden increase in traffic, a slow or sporadic link, or inadequate resources—it will discard incoming IP datagrams. When it does this, the router might send a "Source Quench" message back to the originator of the IP datagram telling the originator that datagrams are arriving too quickly to be processed. The destination node can also send "Source Quench" messages back to the originator for similar reasons. This isn't done for each datagram discarded; rather, it is done for segments of messages or not all. The Internet Engineering Task Force's (IETF) RFC 1812 recommends that "Source Quench" messages not be sent at all because they create more traffic on an already crowded circuit. If "Source Quench" messages are received, however, the originator will resend the related TCP segment at a slower transmission rate to avoid the congestion.

When subnetting is used, the first part of the IP address cannot be used

When subnetting is used, the first part of the IP address cannot be used to determine the subnet mask. To discover its subnet mask, an IP node sends an “Address Request” message to a known router or uses either an all-subnets-directed broadcast or a limited broadcast IP address. A router responding to the message sends an “Address Reply” message, which contains the subnet mask for the network segment on which the “Address Request” message was received. If an IP node doesn’t know its IP address, it can also send an “Address Request” message with a source IP address of 0.0.0.0. The receiver of this message assumes the source IP node uses a class-based subnet mask and responds accordingly using a broadcast.

Before data is transmitted over TCP, the receiver advertises how much data it can receive at one time. This value is called the TCP window size. When transferring data, the TCP window size determines how much data can be transmitted before the sender has to wait for an acknowledgment from the receiver. The TCP window size is a 16-bit field, allowing for a maximum receive window size of 65,535 bytes. This means that a source node could send up to that amount of data in a single TCP window. Using the TCP window scale option, a receiver can advertise a larger window size of up to approximately 1 gigabyte (GB).

To calculate the retransmission timeout (RTO) value to use, TCP tracks the round-trip time (RTT) between TCP segments on an ongoing basis. Normally, the RTO is calculated once for every full window of data sent. In many network environments, this approach works well and prevents having to retransmit data. However, in a high-bandwidth environment or if there are long delays in any environment, this technique doesn’t work so well. One sampling of data for each window cannot be used to determine the current RTO correctly and prevent unnecessary retransmission of data.

To allow for calculating the RTT and in turn the RTO on any TCP segment, a timestamp value based on the local clock is sent in a Timestamp Request message. The acknowledgment for the data on the TCP segment echoes back the timestamp, which allows the RTT to be calculated using the echoed timestamp and the time that the segment’s acknowledgment arrived. These messages are recorded as Timestamp Requests and Timestamp Replies.

Timestamp Replies.

The final type of ICMP message of import in troubleshooting is the Redirect message. Redirect messages are used to tell the senders of IP datagrams about a more optimal route from the sender to the destination node. Because most hosts maintain minimal routing tables, this information is used to improve message routing while decreasing transfer times and errors. So when you see Redirect messages, you know traffic is being rerouted to a destination node.

Examining Fragmentation, Reassembly, and Error Details

To dig deeper into IP datagram fragmenting and reassembly, you can type **netsh interface ipv4 show ipstats**. The output should look similar to the following:

```
MIB-II IP Statistics
------------------------------------------------------
Forwarding is:    Enabled
Default TTL:    128
In Receives:    24219
In Header Errors:    0
In Address Errors:    250
Datagrams Forwarded:    0
In Unknown Protocol:    0
In Discarded:    0
In Delivered:    23969
Out Requests:    20738
Routing Discards:    0
Out Discards:    0
Out No Routes:    0
Reassembly Timeouts:    60
Reassembly Required    0
Reassembled Ok:    0
Reassembly Failures:    0
Fragments Ok:    0
Fragments Failed:    0
Fragments Created:    0
```

TIP You can refresh these statistics automatically. Add the **Rr=***RefreshRate* parameter, where *RefreshRate* is the number of seconds to use as the refresh interval.

As you can see the IP statistics show the default TTL value for outbound packets created on this computer for transmission. Here, the TTL value is

128. This means that there can be up to 128 links between this computer and the destination computer. If packets use more hops than that, the packets would be discarded and a “Time Exceeded” message would be sent back to the computer.

The In Receives value specifies how many inbound packets have been received. The actual number of packets used is represented by the In Delivered value, and the difference between these values is the result of inbound packets that are:

- Received with errors, designated as either In Header Errors or In Address Errors
- Forwarded to other IP nodes, designated as Datagrams Forwarded
- Using an unknown protocol, designated as In Unknown Protocol
- Discarded, such as when a packet’s TTL is exceeded, designated as In Discarded

In this example there is a 250-datagram difference between the In Receives and In Delivered values, because of 250 inbound packets with addressing errors.

The number and disposition of outbound packets is also recorded. The number of packets being transmitted out is listed as Out Requests. Any errors coming back as a result of those transmissions are recorded according to type. If a router or other node sends back a “Destination Unreachable” message, this is usually recorded as a Routing Discard. Other types of error messages, such as “Parameter Problem” or “Source Quench” messages, might be recorded as Routing Discards or Out Discards. If there is no route out or if a “No Route” message is returned, the packets might be recorded as Out No Routes.

When data is transmitted outside the local network over routers, it is typically fragmented and reassembled as mentioned previously. Statistics for the reassembly of the original datagrams is recorded and so is the status of received fragments.

Examining Current TCP and UDP Connections

Firewalls and proxy servers can affect the ability to connect a system on the local network to systems on remote networks. Typically, an administrator will have to open TCP or UDP ports to allow remote communications between a computer on the local network and the remote computer or network. Each type of application or utility that you use may require different ports to be opened. A complete list of TCP and UDP ports used by well-known services is stored in \%*SystemRoot*%\System32\Drivers\
Etc\Services.

Sometimes, however, the tool you want to work with won't have a well-known service associated with it and you may need to experiment a bit to find out what TCP or UDP ports it works with. One way to do this is to start the tool and use a TCP or UDP listener to see what ports become active.

Working with TCP

TCP ports are made available using a passive open, which basically says the port is available to receive requests. When a client wants to use an available port, it must try to establish a connection. A TCP connection is a two-way connection between two clients using Application Layer protocols on an IP network. The TCP endpoints are identified by an IP address and TCP port pair. There is a local TCP endpoint and a remote TCP endpoint, which can be used to identify loopback connections from the local computer to the local computer as well as standard connections from the local computer to a remote computer somewhere on the network. TCP connections are established using a three-way handshake. Here's how that works: 1. A client wanting to use a port sends an active open request (SYN).

2. The local client acknowledges the request, sending a SYN-ACK.
3. To which the client wanting to use the port sends a final acknowledgment (ACK).

Data passed over a TCP connection is apportioned into segments. Segments are sent as IP datagrams that contain a TCP header and TCP data. When a connection is established, the maximum segment size (MSS) is also set. Typically, the maximum value for the MSS is 65,495 bytes, which is 65,535 bytes for the IP datagram minus the minimum IP header (20 bytes) and the minimum TCP header (20 bytes). Technically speaking, SYN, SYN-ACK, and ACK messages are SYN, SYN-ACK, and ACK segments.

You can view current TCP connection statistics by typing **netsh interface ipv4 show tcpconn**. Refresh the statistics automatically by adding the Rr=*RefreshRate* parameter. The output shows you which TCP ports are being listened on, which TCP ports have established connections, and which ports are in a wait state, as shown in this example:

```
MIB-II TCP
Connection Entry
Local Address Local Port Remote Address     Remote Port   State
---------------------------------------------------------------
0.0.0.0               42   0.0.0.0          18520        Listen
0.0.0.0               53   0.0.0.0          16499        Listen
0.0.0.0               88   0.0.0.0          45165        Listen
0.0.0.0              135   0.0.0.0           2176        Listen
0.0.0.0              389   0.0.0.0           2256        Listen
0.0.0.0             1025   0.0.0.0          43054        Listen
127.0.0.1            389   127.0.0.1         1033     Established
127.0.0.1            389   127.0.0.1         1034     Established
127.0.0.1            389   127.0.0.1         1035     Established
127.0.0.1            389   127.0.0.1         1039     Established
127.0.0.1           1033   127.0.0.1          389     Established
192.168.1.50        3289   192.168.1.50       135          Wait
192.168.1.50         290   192.168.1.50      1025          Wait
```

Entries for 0.0.0.0 represent TCP broadcasts. Entries for 127.0.0.1 represent local loopback ports used by the local computer. You'll also see entries on the physical IP address used by the computer back to the computer. Here, these are shown with the local and remote IP address set to 192.168.1.50. The entries you are most interested in are those in which the remote IP address is different from the local IP address; these represent connections to other systems and networks.

The Local Port and Remote Port columns show you how local TCP ports are mapped to remote TCP ports. For example, in this output local port 135 on IP address 192.168.1.50 is mapped to the remote port 1040 on IP address 192.168.1.56. Each TCP connection also has a state. The most common state values are summarized in Table 17-3.

TABLE 17-3 TCP Connection States

STATE	**DESCRIPTION**
Closed	No TCP connection currently exists.
Listen	An Application Layer protocol has issued a passive open function call to permit incoming connection requests on the specified port. This doesn’t create any TCP traffic.
Syn Sent	A client using an Application Layer protocol has issued an active open function call (SYN), which creates and sends the first segment of the TCP three-way handshake.
Syn Rcvd	A client using an Application Layer protocol has received the SYN and sent back an acknowledgment (SYN-ACK).
Established	The final ACK has been received and the TCP connection is established. Data can be transferred in both directions.
Wait	The TCP connection has been terminated and this has been acknowledged by both the local and remote client (FIN-ACK).

You can view additional TCP statistics by typing **netsh interface ipv4 show tcpstats**. The output should be similar to the following:

```
MIB-II TCP Statistics
------------------------------------------------------
Timeout Algorithm:                    Van Jacobson's Algorithm
Minimum Timeout:                      10
Maximum Timeout:                      4294967295
Maximum Connections:                  Dynamic
Active Opens:                         182
Passive Opens:                        174
```

```
Attempts Failed:                    6
Established Resets:                 226
Currently Established:              46
In Segments:                        410814
Out Segments:                       410448
Retransmitted Segments:             2811
In Errors:                          0
Out Resets:                         171
```

The TCP statistics detail the following:

- Minimum and maximum timeout values in use
- Total number of active and passive opens since TCP/IP networking was started on the computer
- Any connections that were attempted but failed
- Any connections that were established and then reset
- The total number of connections currently established
- The number of TCP segments sent (in segments) and received (out segments)
- The number of segments that had to be retransmitted
- The number of segments with errors that were received (in errors)

Working with UDP

Unlike TCP, which is connection-oriented, UDP is connectionless, meaning that UDP messages are sent without negotiating a connection. UDP ports are completely separate from TCP ports, even for the same port number. Because UDP messages are sent without sequencing or acknowledgment, they are unreliable, in stark contrast to TCP, which is very reliable. When you work with UDP, you have only a local address and local port pair, which represent the ports being listened on. You can view the related listener entries by typing **netsh interface ipv4 show udpconn**. The output will be similar to the following:

```
MIB-II UDP Listener Entry
Local Address     LocalPort
---------------------------
    0.0.0.0        42
    0.0.0.0        445
    0.0.0.0        500
    0.0.0.0        1030
    0.0.0.0        1032
    0.0.0.0        1701
    0.0.0.0        3002
    0.0.0.0        3103
    0.0.0.0        3114
    0.0.0.0        4500
    127.0.0.1        53
    127.0.0.1        123
    127.0.0.1        1036
    127.0.0.1        3101
    127.0.0.1        3102
    192.168.1.50        53
    192.168.1.50        67
    192.168.1.50        137
    192.168.1.50        138
    192.168.1.50        389
    192.168.1.50        464
    192.168.1.50            2535
```

Entries for 0.0.0.0 represent UDP broadcast ports. Entries for 127.0.0.1 represent local loopback ports used by the local computer. Entries on the physical IP address for a network adapter are ports which are listening for connections.

UDP messages are sent as IP datagrams and consist of a UDP header and UDP message data. You can view additional UDP statistics by typing **netsh interface ipv4 show udpstats**. The output should be similar to the following:

```
MIB-II UDP Statistics
---------------------
In Datagrams:    42640
In Invalid Port:    732
In Erroneous Datagrams:    20
Out Datagrams:    72217
```

The UDP statistics detail the following information:

- The total number of datagrams received on UDP ports
- The number of datagrams received on invalid ports and discarded
- The number of erroneous datagrams that were received and

discarded
- The number of datagrams sent over UDP ports

Troubleshooting TCP/IP Networking

Problems with TCP/IP networking can be difficult to track down, which is why so many tools exist to help you try to determine what's happening. As you start to troubleshoot, make sure you have a clear understanding of the concepts and procedures discussed in the section titled "Supporting TCP/IP Networking" earlier in this chapter. The tools and techniques discussed there will help you uncover and diagnose some of the most complex TCP/IP networking problems. In addition to that discussion, you can use the discussion in this section to troubleshoot connectivity and configuration issues.

Viewing Diagnostic Information

Many TCP/IP networking problems relate to incorrect configuration of networking components and you'll find that one of the fastest and easiest ways to get a complete snapshot of a computer's network configuration and workload is to use the commands available in the Netsh Ras Diagnostics context. Although these commands are meant for troubleshooting remote access, they are also useful in troubleshooting networking in general.

Netsh Ras Diagnostics provides different Show commands that allow you to work with specified types of diagnostics information and different Set commands to allow you to configure diagnostic logging, tracing, and reporting. However, rather than working with individual diagnostics areas, you'll often want to have Netsh Ras Diagnostics generate a comprehensive report for all diagnostics areas. You can create a comprehensive report for diagnostics by typing **netsh ras diagnostics show all type= file destination= *FileName* verbose= enabled** where *FileName* is the name of the HTML file to generate.

If you then view the report in Internet Explorer, you'll see the contents of several logs, including remote access trace logs, modem tracing logs, Connection Manager logs, IPsec tracing logs, and the Remote Access event logs. You'll also see an installation check of components required for Remote Access and networking in general, a detailed listing of all installed networking components, and the current values of all registry keys used with Remote Access. But the information you want to focus on is the output from the command-line utilities, including:

- arp.exe -a, which displays Address Resolution Protocol entries.
- ipconfig.exe /all, which displays the configuration of all network interfaces.
- ipconfig.exe /displaydns, which displays the contents of the DNS resolver cache.
- route.exe print, which prints contents of the IPv4 and IPv6 routing tables.
- net.exe start, which lists all running Windows services.
- netstat.exe -e, which lists network interface statistics for packet transmissions.
- netstat.exe -o, which lists active connections by protocol, including those for both TCP and UDP connections.
- netstat.exe -s, which lists summary statistics for IPv4, IPv6, ICMPv4, ICMPv6, TCP for IPv4, TCP for IPv6, UDP for IPv4, and UDP for IPv6.
- netstat.exe -n, which lists active connections by address and port, including those for both TCP and UDP connections.
- nbtstat.exe -c, which lists the contents of the cache for NetBIOS over TCP/IP.
- nbtstat.exe -n, which lists local NetBIOS names.
- nbtstat.exe -r, which lists names resolved by broadcast and WINS.
- nbtstat.exe -S, which lists NetBIOS session tables with the destination IP addresses.
- netsh.exe dump, which performs a complete dump of the

network configuration. You can use this dump to review the network configuration or to recreate the network configuration in the future.

NOTE While you could run each of these commands yourself, don't overlook how useful Netsh can be to remotely troubleshoot problems. With Netsh, you don't need to sit at the user's computer or log on remotely using Remote Desktop. You simply start Netsh with the –R parameter to provide the name of the remote computer you want to work with and then go about diagnosing the problem at hand.

Diagnosing General Computer Configuration Issues

As part of diagnostic troubleshooting, you may want to view detailed configuration information for the computer and the operating system, and one of the best ways to do this is to access the Windows Management Instrumentation (WMI) objects that track the related configuration settings. The WMI object that tracks overall computer configuration is the Win32_ComputerSystem object. The WMI object that tracks overall operating system configuration is the Win32_OperatingSystem object.

One way to work with WMI is to use Windows PowerShell. In Windows PowerShell, you can use the Get-WMIObject cmdlet to get the WMI object you want to work with. By redirecting the object to Format-List * you can list all of the properties of the object and their values. When working with WMI, you'll want to work with the root namespace, as specified by setting the –Namespace parameter to root/cimv2. Using the –Computer parameter, you can specify the computer you want to work with. If you want to work with the local computer, use a dot (.) instead of a computer name.

You can thus examine the Win32_ComputerSystem object and its properties to obtain summary information regarding a computer's configuration by entering the following command at the Windows

PowerShell prompt: `Get-WmiObject -Class Win32_ComputerSystem -Namespace root/cimv2 -ComputerName . | Format-List *`

If you want to save the output in a file, simply redirect the output to a file. In the following example, you redirect the output to a file in the working directory called computer_save.txt: `Get-WmiObject -Class Win32_ComputerSystem -Namespace root/cimv2 -ComputerName . | Format-List * > computer_save.txt` The detailed computer information, obtained by using this command, is shown in Listing 17-2.

LISTING 17-2 Computer Configuration Information

```
PSComputerName              : CORPSERVER64
AdminPasswordStatus         : 0
BootupState                 : Normal boot ChassisBootupState          :
3
KeyboardPasswordStatus      : 0
PowerOnPasswordStatus       : 0
PowerSupplyState            : 3
PowerState                  : 0
FrontPanelResetStatus       : 0
ThermalState                : 3
Status                      : OK
Name                        : CORPSERVER64
PowerManagementCapabilities : PowerManagementSupported    :
__GENUS                     : 2
__CLASS                     : Win32_ComputerSystem
__SUPERCLASS                : CIM_UnitaryComputerSystem
__DYNASTY                   : CIM_ManagedSystemElement
__RELPATH                   : Win32_ComputerSystem.Name="CORPSERVER64"
__PROPERTY_COUNT            : 60
__DERIVATION                : {CIM_UnitaryComputerSystem,
CIM_ComputerSystem, CIM_System, CIM_LogicalElement...}
__SERVER                    : CORPSERVER64
__NAMESPACE                 : root\cimv2
__PATH                      :
\\CORPSERVER64\root\cimv2:Win32_ComputerSystem.Name="CORPSERVER64"
AutomaticManagedPagefile    : True AutomaticResetBootOption    : True
```

```
AutomaticResetCapability    : True BootOptionOnLimit           :
BootOptionOnWatchDog        : BootROMSupported            : True
Caption                     : CORPSERVER64
CreationClassName           : Win32_ComputerSystem
CurrentTimeZone             : -480
DaylightInEffect            : False Description                : AT/AT
COMPATIBLE
DNSHostName                 : CorpServer64
Domain                      : imaginedlands.local
DomainRole                  : 5
EnableDaylightSavingsTime   : True HypervisorPresent           : False
InfraredSupported           : False InitialLoadInfo             :
InstallDate                 : LastLoadInfo                :
Manufacturer                : Dell Inc.
Model                       : OptiPlex 790
NameFormat                  : NetworkServerModeEnabled    : True
NumberOfLogicalProcessors   : 4
NumberOfProcessors          : 1
OEMLogoBitmap               : OEMStringArray              : {Dell
System, 1[04AD], 3[1.0], 12[www.dell.com]}
PartOfDomain                : True PauseAfterReset             : -1
PCSystemType                : 1
PCSystemTypeEx              : 1
PrimaryOwnerContact         : PrimaryOwnerName            : Windows User
ResetCapability             : 1
ResetCount                  : -1
ResetLimit                  : -1
Roles                       : {LM_Workstation, LM_Server,
Primary_Domain_Controller, Timesource...}
SupportContactDescription   : SystemStartupDelay          :
SystemStartupOptions        : SystemStartupSetting        :
SystemType                  : x64-based PC
TotalPhysicalMemory         : 12760170496
UserName                    : WakeUpType                  : 6
Workgroup                   : Scope                       :
System.Management.ManagementScope Path                       :
\\CORPSERVER64\root\cimv2:Win32_ComputerSystem.Name="CORPSERVER64"
```

```
Options                     : System.Management.ObjectGetOptions
ClassPath                   :
\\CORPSERVER64\root\cimv2:Win32_ComputerSystem
Properties                  : {AdminPasswordStatus,
AutomaticManagedPagefile, AutomaticResetBootOption,
AutomaticResetCapability...}
SystemProperties            : {__GENUS, __CLASS, __SUPERCLASS,
__DYNASTY...}
Qualifiers                  : {dynamic, Locale, provider, UUID}
Site                        : Container
```

: A summary of the computer configuration entries and their meaning is provided in Table 17-4.

TABLE 17-4 Computer Configuration Entries and Their Meaning

PROPERTY	DESCRIPTION
AdminPasswordStatus	Status of the Administrator password. Values are: 1 = Disabled, 2 = Enabled, 3 = Not Implemented, 4 = Unknown.
AutomaticManagedPagefile	Indicates whether the computer's page file is being managed by the operating system.
AutomaticResetBootOption	Indicates whether the automatic reset boot option is enabled.
AutomaticResetCapability	Indicates whether the automatic reset is enabled.
BootOptionOnLimit	System action to be taken when the ResetLimit value is reached. Values are: 1 = Reserved, 2 = Operating system, 3 = System utilities, 4 = Do not reboot.
BootOptionOnWatchDog	Reboot action to be taken after the time on the watchdog timer has elapsed. Values are: 1 = Reserved, 2 = Operating system, 3 = System utilities, 4 = Do not reboot.

BootROMSupported

Indicates whether a boot ROM is supported.

BootupState

Indicates how the system was started. Values are: “Normal boot”, “Fail-safe boot”, and “Fail-safe with network boot”.

Caption

System name.

ChassisBootupState

Bootup state of the system chassis. Values are: 1 = Other, 2 = Unknown, 3 = Safe, 4 = Warning, 5 = Critical, 6 = Nonrecoverable.

ClassPath

The Windows Management Instrumentation (WMI) object class path.

Container

The container associated with the object.

CreationClassName

Name of class from which object is derived.

CurrentTimeZone

Number of minutes the computer is offset from Coordinated Universal Time.

DaylightInEffect

Indicates whether daylight savings mode is on.

Description

Description of the computer.

DNSHostName

Name of the server according to DNS.

Domain

Name of the domain to which the computer belongs.

DomainRole

Domain role of the computer. Values are:
0 = Standalone Workstation,
1 = Member Workstation,
2 = Standalone Server, 3 = Member Server,
4 = Backup Domain Controller,
5 = Primary Domain Controller.

EnableDaylightSavingsTime

Indicates whether Daylight Savings Time is enabled. If TRUE, the system changes to an hour ahead or behind when DST starts or ends. If FALSE, the system does not change to an hour ahead or behind when DST starts or ends.

FrontPanelResetStatus

Hardware security settings for the reset button on the computer. Values are: 0 = Disabled,
1 = Enabled, 2 = Not Implemented,
3 = Unknown.

InfraredSupported

Indicates whether an infrared (IR) port exists on the computer system.

InitialLoadInfo

Data needed to find either the initial load device (its key) or the boot service to request the operating system to start up.

InstallDate

When the computer was installed.

KeyboardPasswordStatus

Indicates the keyboard password status. Values are: 0 = Disabled, 1 = Enabled,
2 = Not Implemented, 3 = Unknown.

LastLoadInfo

Array entry of the InitialLoadInfo property, which holds the data corresponding to booting the currently loaded operating system.

Manufacturer

Computer manufacturer name.

Model

Product name given by the manufacturer.

Name

The computer name.

NameFormat

Identifies how the computer system name is generated.

NetworkServerModeEnabled

Indicates whether Network Server Mode is enabled.

NumberOfLogicalProcessors

Number of processor cores. If the computer has two processors with four cores each, the number of logical processors is eight. If the computer has hyperthreading architecture, the number of logical processors may also be higher than the number of physical processors.

NumberOfProcessors

Number of enabled processors on the computer.

OEMLogoBitmap

Identifies the bitmap for the OEM's logo.

OEMStringArray

List of descriptive strings set by the OEM.

PartOfDomain

Indicates whether the computer is part of a domain. If TRUE, the computer is a member of a domain. If FALSE, the computer is a member of a workgroup.

Options

Lists the management object options.

Path

Identifies the full WMI path to the object class.

PauseAfterReset

Time delay in milliseconds before a reboot is initiated after a system power cycle or reset. A value of -1 indicates there is no time delay.

PCSystemType

Indicates the type of computer. Values are:
0 = Unspecified, 1 = Desktop, 2 = Mobile,
3 = Workstation, 4 = Enterprise Server,
5 = SOHO Server, 6 = Appliance PC,
7 = Performance Server, 8 = Role Maximum.

PowerManagementCapabilities Power management capabilities of a logical device. Values are: 0 = Unknown,
1 = Not Supported,2 = Disabled, 3 = Enabled,
4 = Power Saving Modes Entered Automatically, 5 = Power State Settable,
6 = Power Cycling Supported,
7 = Timed Power On Supported.

PowerManagementSupported

Indicates whether the device’s power can be managed.

PowerOnPasswordStatus

Power on password status. Values are:
0 = Disabled, 1 = Enabled,
2 = Not Implemented, 3 = Unknown.

PowerState

Indicates the current power state of the computer.

Values are: 0 = Unknown, 1 = Full Power,
2 = Power Save – Low Power Mode,
3 = Power Save – Standby,
4 = Power Save – Unknown, 5 = Power Cycle,
6 = Power Off, 7 = Power Save – Warning.

PowerSupplyState

State of the enclosure’s power supply when last booted. Values are: 1 = Other, 2 = Unknown,
3 = Safe, 4 = Warning, 5 = Critical,
6 = Nonrecoverable.

PrimaryOwnerContact

Contact information for the computer’s owner.

PrimaryOwnerName

Name of the system owner.

Properties

Lists all the properties of the object.

Qualifiers

Lists any qualifiers for the object.

ResetCapability

Value indicates whether a computer can be reset using the Power and Reset buttons (or other hardware means). Values are: 1 = Other,
2 = Unknown, 3 = Disabled, 4 = Enabled,
5 = Nonrecoverable.

ResetCount

Number of automatic resets since the last intentional reset. A value of -1 indicates that the count is unknown.

ResetLimit

Number of consecutive times a system reset will be attempted. A value of -1 indicates that the limit is unknown.

Roles

System roles.

Scope

Lists the management object scope.

Site

The site associated with the object.

Status

Current status of the computer. Values are: “OK”, “Error”, “Degraded”, “Unknown”, “Pred Fail”, “Starting”, “Stopping”, “Service”.

SupportContactDescription

List of the support contact information for the computer.

SystemProperties

Lists the system properties.

SystemStartupDelay

The startup delay in seconds.

SystemStartupOptions

List of the startup options for the computer.

SystemStartupSetting

Index of the default start profile.

SystemType

System architecture type, such as “X86-based PC” or “64-bit Intel PC”.

ThermalState

Thermal state of the system chassis when last booted. Values are: 1 = Other, 2 = Unknown,
3 = Safe, 4 = Warning, 5 = Critical,
6 = Nonrecoverable.

TotalPhysicalMemory

Total byte size of physical memory.

UserName

Name of the currently logged-on user.

WakeUpType

Event that caused the system to power up. Values are: 0 = Reserved, 1 = Other,
2 = Unknown, 3 = APM Timer, 4 = Modem Ring, 5 = LAN Remote, 6 = Power Switch,
7 = PCI PME#, 8 = AC Power Restored.

Workgroup

When a computer is a member of a workgroup, the workgroup name is listed here.

As you can see, the detailed configuration information tells you a great deal about the computer’s configuration. The same is true for the operating system details, which can be obtained by entering the following command at a Windows PowerShell prompt: `Get-WmiObject -Class Win32_OperatingSystem -Namespace root/cimv2 -ComputerName . | Format-List *`

Listing 17-3 provides an example of the output from this command. As

Listing 17-3 provides an example of the output from this command. As discussed previously, you can redirect the output to a save file.

LISTING 17-3 Verbose Operating System Configuration Output

```
PSComputerName                            : CORPSERVER64
Status                                    : OK
Name                                      : Microsoft Windows Server
2012 R2 Datacenter|C:\Windows|\Device\Harddisk0\Partition2
FreePhysicalMemory                        : 10210800
FreeSpaceInPagingFiles                    : 1900544
FreeVirtualMemory                         : 12038188
__GENUS                                   : 2
__CLASS                                   : Win32_OperatingSystem
__SUPERCLASS                              : CIM_OperatingSystem
__DYNASTY                                 : CIM_ManagedSystemElement
__RELPATH                                 : Win32_OperatingSystem=@
__PROPERTY_COUNT                          : 64
__DERIVATION                              : {CIM_OperatingSystem,
CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                                  : CORPSERVER64
__NAMESPACE                               : root\cimv2
__PATH                                    :
\\CORPSERVER64\root\cimv2:Win32_OperatingSystem=@
BootDevice                                : \Device\HarddiskVolume1
BuildNumber                               : 9600
BuildType                                 : Multiprocessor Free
Caption                                   : Microsoft Windows Server
2012 R2 Datacenter CodeSet                                  : 1252
CountryCode                               : 1
CreationClassName                         : Win32_OperatingSystem
CSCreationClassName                       : Win32_ComputerSystem
CSDVersion                                :
CSName                                    : CORPSERVER64
CurrentTimeZone                           : -480
DataExecutionPrevention_32BitApplications : True
DataExecutionPrevention_Available         : True
DataExecutionPrevention_Drivers           : True
DataExecutionPrevention_SupportPolicy     : 3
Debug                                     : False
Description                               :
Distributed                               : False
EncryptionLevel                           : 256
```

```
ForegroundApplicationBoost                  : 2
InstallDate                                 : 20140812194435.000000-420
LargeSystemCache                            :
LastBootUpTime                              : 20150109131130.490920-480
LocalDateTime                               : 20150109161406.091000-480
Locale                                      : 0409
Manufacturer                                : Microsoft Corporation
MaxNumberOfProcesses                        : 4294967295
MaxProcessMemorySize                        : 137438953344
MUILanguages                                : {en-US}
NumberOfLicensedUsers                       :
NumberOfProcesses                           : 50
NumberOfUsers                               : 7
OperatingSystemSKU                          : 8
Organization                                :
OSArchitecture                              : 64-bit
OSLanguage                                  : 1033
OSProductSuite                              : 400
OSType                                      : 18
OtherTypeDescription                        :
PAEEnabled                                  :
PlusProductID                               :
PlusVersionNumber                           :
PortableOperatingSystem                     : False
Primary                                     : True
ProductType                                 : 2
RegisteredUser                              : Windows User
SerialNumber                                : 00252-80525-19743-AA445
ServicePackMajorVersion                     : 0
ServicePackMinorVersion                     : 0
SizeStoredInPagingFiles                     : 1900544
SuiteMask                                   : 400
SystemDevice                                : \Device\HarddiskVolume2
SystemDirectory                             : C:\Windows\system32
SystemDrive                                 : C:
TotalSwapSpaceSize                          :
TotalVirtualMemorySize                      : 14361648
TotalVisibleMemorySize                      : 12461104
Version                                     : 6.3.9600
WindowsDirectory                            : C:\Windows
Scope                                       :
System.Management.ManagementScope
```

```
Path                                        :
\\CORPSERVER64\root\cimv2:Win32_OperatingSystem=@
Options                                     :
System.Management.ObjectGetOptions
ClassPath                                   :
\\CORPSERVER64\root\cimv2:Win32_OperatingSystem
Properties                                  : {BootDevice, BuildNumber,
BuildType, Caption...}
SystemProperties                            : {__GENUS, __CLASS,
__SUPERCLASS, __DYNASTY...}
Qualifiers                                  : {dynamic, Locale, provider,
Singleton...}
Site                                        :
Container                                   :
```

A summary of the operating system entries and their meanings is provided in Table 17-5.

TABLE 17-5 Operating System Configuration Entries and Their Meanings

PROPERTY	DESCRIPTION
BootDevice	Disk drive from which the Win32 operating system boots.
BuildNumber	Build number of the operating system.
BuildType	Type of build used for the operating system, such as "retail build", "checked build", or "multiprocessor free".
Caption	Operating system name.
CodeSet	Code page value used by the operating system.
Container	The container associated with the object.
CountryCode	

Country code used by the operating system.

CreationClassName

Name of class from which the object is derived.

CSCreationClassName

Name of class from which computer system object is derived.

CSDVersion

Indicates the latest Service Pack installed on the computer. Value is NULL if no Service Pack is installed.

CSName

Name of the computer system associated with this object class.

CurrentTimeZone

Number of minutes the operating system is offset from Greenwich Mean Time. The value is positive, negative, or zero.

DataExecutionPrevention_32BitApplications Indicates whether Data Execution Prevention (DEP) is enabled for 32-bit applications.

DataExecutionPrevention_Available Indicates whether Data Execution Prevention (DEP) is supported by the system hardware.

DataExecutionPrevention_Drivers Indicates whether Data Execution Prevention (DEP) is enabled for device drivers.

DataExecutionPrevention_SupportPolicy Specifies the DEP support policy being used. Values are: 0 = none, 2 = on for essential Windows programs and services only, 3 = on for all programs except those specifically excluded.

Debug

Indicates whether the operating system is a checked (debug) build. If TRUE, the debugging version of User.exe is installed.

Description

Description of the Windows operating system.

Distributed

Indicates whether the operating system is distributed across multiple computer system nodes. If so, these nodes should be grouped as a cluster.

EncryptionLevel

The level of encryption for secure transactions as 40-bit, 128-bit, or n-bit.

ForegroundApplicationBoost Sets the priority of the foreground application. Application boost is implemented by giving an application more processor time. Values are:
0 = None, 1 = Minimum, 2 = Maximum (Default).

FreePhysicalMemory

Physical memory in kilobytes currently unused and available.

FreeSpaceInPagingFiles

Amount of free space in kilobytes in the operating system's paging files. Swapping occurs when the free space fills up.

FreeVirtualMemory

Virtual memory in kilobytes unused and available.

InstallDate

When the operating system was installed.

LargeSystemCache

Indicates whether memory usage is optimized for program or the system cache. Values are:
0 = memory usage is optimized for programs,
1 = memory usage is optimized for the system cache.

LastBootUpTime

When the operating system was last booted.

LocalDateTime

Local date and time on the computer.

Locale

Language identifier used by the operating system.

Manufacturer

Operating system manufacturer. For Win32 systems, this value will be "Microsoft Corporation".

MaxNumberOfProcesses

Maximum number of process contexts the operating system can support. If there is no fixed maximum, the value is 0.

MaxProcessMemorySize

Maximum memory in kilobytes that can be allocated to a process. A value of zero indicates that there is no maximum.

MUILanguages

User interface languages supported.

Name

Name of the operating system instance.

NumberOfLicensedUsers

Number of user licenses for the operating system. A value of 0 = unlimited, a value of –1 = unknown.

NumberOfProcesses

Current number of process contexts on the system.

NumberOfUsers

Current number of user sessions.

OperatingSystemSKU

Operating system product type indicator.

Options

Lists the management object options.

Organization

Company name set for the registered user of the operating system.

OSArchitecture

The operating system architecture as 32-bit or 64-bit.

OSLanguage

Language version of the operating system installed.

OSProductSuite

Operating system product suite installed.

OSType

Type of operating system. Values include:
1 = Other, 18 = Windows NT or later.

OtherTypeDescription

Sets additional description; used when OSType = 1.

Path

Identifies the full WMI path to the object class.

PAEEnabled

Indicates whether physical address expansion (PAE) is enabled.

PlusProductID

Product number for Windows Plus! (if installed).

PlusVersionNumber

Version number of Windows Plus! (if installed).

Primary

Indicates whether this is the primary operating system.

ProductType

The operating system product type. Values are:
1 = workstation, 2 = domain controller,
3 = server.

Properties

Lists all the properties of the object.

RegisteredUser

Name set for the registered user of the operating system.

Scope

Lists the management object scope.

SerialNumber

Operating system product serial number.

ServicePackMajorVersion

Major version number of the service pack installed on the computer. If no service pack has been installed, the value is zero or NULL.

ServicePackMinorVersion

Minor version number of the service pack installed on the computer. If no service pack has been installed, the value is zero or NULL.

Site

The site associated with the object.

Status

Current status of the object. Values include: “OK”, “Error”, “Unknown”, “Degraded”, “Pred Fail”, “Starting”, “Stopping”, and “Service”.

SuiteMask

Bit flags that identify the product suites available on the system.

SystemDevice

Physical disk partition on which the operating system is installed.

SystemDirectory

System directory of the operating system.

SystemDrive

The physical disk partition on which the operating system is installed.

TotalSwapSpaceSize

Total swap space in kilobytes. This value may be unspecified (NULL) if swap space is not distinguished from page files.

TotalVirtualMemorySize

Virtual memory size in kilobytes.

TotalVisibleMemorySize

Total amount of physical memory in kilobytes that is available to the operating system.

Version

Version number of the operating system.

WindowsDirectory

Windows directory of the operating system.

Diagnosing IP, DNS, and WINS Configuration Issues

The Netsh Interface IPv4 context provides commands for viewing the IP, DNS, and WINS configuration on a computer. These commands, with example output, are as follows:

- **Netsh interface ipv4 show addresses** Shows the IP addresses used by network adapters on the computer, as in the following example:

```
Configuration for interface "Local Area Connection"
    DHCP enabled:                        Yes
    IP Address:                          192.168.1.101
    Subnet Prefix:                       192.168.1.0/24 (mask
255.255.255.0)
    Default Gateway:                     192.168.1.1
    Gateway Metric:                      0
    InterfaceMetric:                     5

Configuration for interface "Local Area Connection 2"
    DHCP enabled:                        Yes
    IP Address:                          192.168.5.42
    Subnet Prefix:                       192.168.51.0/24 (mask
255.255.255.0)
    Default Gateway:                     192.168.5.1
    Gateway Metric:                      0
    InterfaceMetric:                     5

Configuration for interface "Loopback Pseudo-Interface 1"
```

```
    DHCP enabled:                         No
    IP Address:                           127.0.0.1
    Subnet Prefix:                        127.0.0.0/8 (mask 255.0.0.0)
    InterfaceMetric:                      50
```

Each network adapter is listed in order. Because this computer has two network adapters, there are two entries in addition to the entry for the loopback interface. Any network adapter that is disabled or otherwise unavailable won't be listed.

Each gateway is listed on a per-adapter basis in the order it is used. If a computer has multiple network adapters, each network adapter that is configured and used should have an entry. There is no notation for an incorrectly configured gateway (that is, one that isn't on the same subnet).

- **Netsh interface ipv4 show dnsservers** Shows the DNS servers defined for network adapters on the computer, as in the following example:

```
Configuration for interface "Local Area Connection"
    DNS servers configured through DHCP:  192.168.1.120
                                          192.168.1.225
    Register with which suffix:           Primary only

Configuration for interface "Local Area Connection2"
    DNS servers configured through DHCP:  192.168.5.86
                                          192.168.5.124
    Register with which suffix:           Primary only

Configuration for interface "Loopback Pseudo-Interface 1"
    Statically Configured DNS Servers:    None
    Register with which suffix:           Primary only
```

Each DNS server configured is shown in the search order used. Confirm that the correct IP addresses are used and that the search order is correct.

- **Netsh interface ipv4 show winsservers** Shows the WINS servers defined for network adapters on the computer, as in the following example:

```
Configuration for interface "Local Area Connection"
    WINS servers configured through DHCP: 192.168.1.128
                                          192.168.1.144
```

```
Configuration for interface "Local Area Connection 2"
    WINS servers configured through DHCP: 192.168.5.45
                                          192.168.5.67

Configuration for interface "Loopback Pseudo-Interface 1"
    Statically Configured WINS Servers:   None
```

Each WINS server configured is shown in the search order used. Confirm that the correct IP addresses are used and that the search order is correct.

As discussed previously, you can also type **Ipconfig /all** at a command prompt to display all TCP/IP configuration information at once. With DHCPv4, you can release and then renew the IPv4 configuration by typing the following commands at a command prompt:

```
ipconfig /release
ipconfig /renew
```

With DHCPv6, you can release and then renew the IPv6 configuration by entering the following commands at a command prompt:

```
ipconfig /release6
ipconfig /renew6
```

If you suspect a computer has problems with DNS, you may also want to display the contents of the DNS resolver cache by typing **ipconfig /displaydns** at a command prompt. You can purge the contents of the DNS resolver cache by typing **ipconfig /flushdns** at a command prompt. To refresh all DHCP leases and re-register DNS names, type **ipconfig /registerdns** at a command prompt.

Other commands you can use for troubleshooting TCP/IP configuration and connectivity include the following:

- **Tracert** Traces connections along the path between computers
- **Ping** Determines whether a network connection can be established
- **Pathping** Combines features of Tracert and Ping to trace routes and provide packet loss information

To check connectivity, you can use any of these commands. In the event of a connectivity problem, the output will confirm this. In the following example, the computer is unable to connect to the specified IP address:

```
Pinging 192.168.1.1 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Here, the computer might not have connectivity to the network or the network configuration may be incorrect. Keep in mind that Windows Firewall and other firewalls on both the source computer and the destination node may block these activities.

To see how you can check connections to specific hosts, consider the following examples: Trace the route to 192.168.1.100:

```
tracert 192.168.1.100
```

Try to establish a connection to Mailsever23 by its host name: `ping mailserver23`

Trace the route to Mailserver23.imaginedlands.com and provide packet loss information: `pathping mailserver23.imaginedlands.com` To attempt to verify connectivity to various remote hosts and the TCP/IP configuration, you can do one of the following:

- **Ping the IP address, computer name and fully qualified domain name of a host** Verifies connectivity and name resolution with regard to a remote host according to IP address, computer name, or fully qualified domain name
- **Ping the computer's local loopback address** Verifies that the network adapter and TCP/IP in general is installed and enabled
- **Ping the DHCP, DNS, and WINS servers** Verifies the DHCP, DNS, and WINS server settings of network adapters
- **Ping gateways** Verifies the default gateways settings of network adapters

When a computer is having possible connectivity or configuration problems, you should immediately ask yourself and verify the following questions:

- **Is the computer connected to the network?** If the computer can connect to one of its default gateways (and that gateway isn't configured on the same IP address as one of the computer's IP addresses), the computer is able to connect to the network. If the computer can't connect to any of its gateways, its network cable may be disconnected or it may have a bad network adapter.
- **Does the computer have a bad network adapter?** If the computer can't connect to any of its gateways, it may have a bad network adapter. Try pinging the local loopback address.

Seeing problems connecting to DHCP, DNS, and WINS is also a concern. You troubleshoot problems connecting to a designated server in a similar way. If the computer can connect to the default gateway but can't get to a DNS, DHCP, or WINS server, the server may be down, the IP address in the configuration may be incorrect, or another interconnection between the computer you are working with and the target server may be down. Don't forget, you can use **netsh interface ipv4 show interfaces** and **netsh interface ipv6 show interfaces** to check adapter connection states.

Appendix A. Essential CommandLine Tools Reference

In this book, I've discussed many command-line tools and scripts. This appendix is intended to provide a quick reference to the syntax and usage of these tools as well as other commands and utilities that you may find helpful. These tools are listed alphabetically by tool name. Unless otherwise noted, these tools work the same way on Windows 2012, Windows Server 2012 R2, and Windows 8.1. In addition, if a tool is not included in multiple operating systems, the source of the tool is listed, such as "Windows 2012 only" for the tools available only on Windows 2012 by default.

ARP

Displays and modifies the IP-to-physical address translation tables used by the address resolution protocol (ARP).

```
arp -a [inet_addr] [-N if_addr] [-v]
arp -d inet_addr [if_addr]
arp -s inet_addr eth_addr [if_addr]
```

ASSOC

Displays and modifies file extension associations.

```
assoc [.ext[=[fileType]]]
```

ATTRIB

Displays and changes file attributes.

```
attrib [+r|-r] [+a|-a] [+s|-s] [+h|-h] [+i|-i]
[[drive:] [path] filename] [/s [/d] [/l]]
```

BCDEDIT

Displays and manages the boot configuration data (BCD) store.

```
bcdedit /command [options]
bcdedit [/v]
```

With Bcdedit, commands you can use include:

- **/bootdebug** Enables or disables boot debugging for a boot application.
- **/bootems** Enables or disables Emergency Management Services for a boot application.
- **/bootsequence** Sets a one-time boot sequence for the boot manager.
- **/copy** Makes copies of entries in the BCD store.
- **/create** Creates new entries in the BCD store.
- **/createstore** Creates a new and empty boot configuration data store.
- **/dbgsettings** Sets global debugger parameters.
- **/debug** Enables or disables kernel debugging for an operating system entry.
- **/default** Sets the default entry for the boot manager.
- **/delete** Deletes entries from the BCD store.
- **/deletevalue** Deletes entry options from the BCD store.
- **/displayorder** Sets the order in which the boot manager displays available operating systems.
- **/ems** Enables or disables Emergency Management Services for an operating system entry.
- **/emssettings** Sets the global Emergency Management Services parameters.
- **/enum** Lists entries in the store.
- **/export** Exports the contents of the system store to a file. This file can be used later to restore the state of the system store.
- **/import** Restores the state of the system store using a backup

file created with the /export command.

- **/set** Sets entry option values in the BCD store.
- **/timeout** Sets a timeout value for the boot manager.
- **/toolsdisplayorder** Sets the order in which the boot manager displays the tools menu.

CACLS

This command is deprecated. See ICACLS.

CALL

Calls a script or script label as a procedure.

```
call [drive:][path]filename [batch–parameters]
call :label [args]
```

CD

Displays the name of or changes the current directory.

```
chdir [/d] [drive:][path]
chdir [..]
cd [/d] [drive:][path]
cd [..]
```

CHDIR

See CD.

CHKDSK

Checks a disk for errors and displays a report.

```
chkdsk [drive:][[path]filename]
[/f][/v][/r][/x][/i][/c][/l[:size]] [/b] [/scan] [/spotfix]
```

CHKNTFS

Displays the status of volumes. Sets or excludes volumes from automatic system checking when the computer is started.

```
chkntfs [/x | /c] volume: [...]
chkntfs /t[:time]
chkntfs /d
chkntfs volume: [...]
```

CHOICE

Creates a selection list from which users can select a choice in batch scripts.

```
choice [/c choices] [/n] [/cs] [/t nnnn /d choice] [/m "text"]
```

CIPHER

Displays current encryption status or modifies folder and file encryption on NTFS volumes.

```
cipher [/e | d | c] [/s:dir] [/b]
       [/h] [[path]filename [...]]
cipher [/k | /r:filename | /w:dir]
cipher u [n]
cipher /x[:efsfile] [filename]
cipher /y
cipher flushcache [SERVER:servername]
cipher /adduser [/certhash:hash | /certfile:filename]
       [/s:dir] [/b] [/h] [pathname [...]]
cipher removeuser certhash:hash
       [/s:dir] [/b] [/h] [pathname [...]]
cipher /rekey [pathname [...]]
```

CLIP

With piping, redirects output of command-line tools to the Windows clipboard.

```
[command |] clip
clip < filename.txt
```

NOTE In this instance, the symbol "|" is the pipe symbol.

CLS

Clears the console window.

```
cls
```

CMD

Starts a new instance of the command shell.

```
cmd [/a | u] [q] [/d] [/e:on | e:off] [t:{bf | f}]
[/f:on | f:off] [v:on | /v:off]
[[/s] [/c | /k] string]
```

CMDKEY

Creates and manages stored user names and passwords.

```
cmdkey [{/add | /generic}:targetname
{/smartcard | /user:user@domain
{/pass{:pwd}}} | /delete{:targetname}
| /ras | /list{:targetname}]
```

COLOR

Sets the colors of the command-shell window.

```
color [[b]f]
```

COMP

Compares the contents of two files or sets of files.

```
comp [data1] [data2] [/d] [/a] [/l]
[/n=number] [/c] [/off[line]]
```

COMPACT

Displays or alters the compression of files on NTFS partitions.

```
compact [/c | u] [s[:dir]] [/a] [/i] [/f]
[/q] [filename [...]]
```

CONVERT

Converts FAT and FAT32 volumes to NTFS.

```
convert volume fs:NTFS [v] [/x]
[/cvtarea:filename] [/nosecurity]
```

COPY

Copies or combines files.

```
copy [/d] [v] [n] [/y|/-y] [/z] [/l] [/a|/b] source [a | b]
[+ source [a | b] [+ ...]][destination [a | b]]
```

DATE

Displays or sets the system date.

```
date [/T | mm-dd-yy]
```

DCDIAG

Performs diagnostics testing on a domain controller.

```
dcdiag [/s:Server[:LDAPPort]] [/u [Domain\]UserName]
[/p {Password | * | ""}] [/h | ?] [xsl] [/a | e] [I] [/fix] [/c]
```

```
[/q | v] [n:NamingContext] [/skip:TestName] [/test:TestName]
[/f:textlogname] [/x:xmllogname]
```

This command applies only to Windows Server 2012 and Windows Server 2012 R2.

DCGPOFIX

Restores default group policy objects.

```
dcgpofix [/ignoreschema]
[/target: {domain | dc | both}]
```

This command applies only to Windows Server 2012 and Windows Server 2012 R2 .

DEFRAG

Defragments hard drives.

```
defrag volume [/a] [/v]
defrag [volume | -c] [{-r | -w}] [-f] [-v]
```

You must run this command using an administrator command prompt.

DEL

Deletes one or more files.

```
del [/p] [/f] [/s] [/q] [/a[[:]attributes]]
[drive:][path]filename[...]
erase [/p] [/f] [/s] [/q] [/a[[:]attributes]] names
```

DIR

Displays a list of files and subdirectories within a directory.

```
dir [drive:][path][filename] [/a[[:]attributes]] [/b] [/c] [/d]
```

```
[/l] [/n] [/o[[:]sortorder]] [/p] [/q] [/r] [/s] [/t[[:]timefield]]
[/w] [/x] [/4]
```

DISKCOMP

Compares the contents of two floppy disks.

```
diskcomp [drive1: [drive2:]]
```

DISKCOPY

Copies the contents of one floppy disk to another.

```
diskcopy [drive1: [drive2:]] [/v]
```

DISKPART

Invokes a text-mode command interpreter so that you can manage disks, partitions, and volumes using a separate command prompt and commands that are internal to DISKPART.

```
diskpart
```

> *MORE INFO* If you run DISKPART in a nonadministrative command prompt, you'll be prompted for consent or permission to run DISKPART in a separate administrative command prompt.

DOSKEY

Edits command lines, recalls Windows commands, and creates macros.

```
doskey [/reinstall] [/listsize=size]
[/macros[:all | :exename]]
[/history] [/insert | /overstrike]
[/exename=exename]
[/macrofile=fname] [macroname=[text]]
```

DRIVERQUERY

Displays a list of all installed device drivers and their properties.

```
driverquery [/s computer [/u [domain\]user [/p [pwd]]]]
[/fo {table|list|csv}] [/nh] [v] [si]
```

DSADD COMPUTER

Creates a computer account in the Active Directory directory service.

```
dsadd computer ComputerDN [-samid SAMName] [-desc Description]
[-loc Location] [-memberof GroupDN ...] [{-s Server | -d Domain}]
[-u UserName] [-p {Password | *}] [-q] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSADD GROUP

Creates a group account in Active Directory.

```
dsadd group GroupDN [-secgrp {yes | no}] [-scope {l | g | u}]
[-samid SAMName] [-desc Description] [-memberof Group ...]
[-members Member ...] [{-s Server | -d Domain}] [-u UserName]
[-p {Password | *}] [-q] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSADD USER

Creates a user account in Active Directory.

```
dsadd user UserDN [-samid SAMName] [-upn UPN] [-fn FirstName]
[-mi Initial]  [-ln LastName] [-display DisplayName]
[-empid EmployeeID] [-pwd {Password | *}] [-desc Description]
[-memberof Group ...] [-office Office] [-tel PhoneNumber]
[-email EmailAddress] [-hometel HomePhoneNumber]
[-pager PagerNumber] [-mobile CellPhoneNumber] [-fax FaxNumber]
[-iptel IPPhoneNumber] [-webpg WebPage] [-title Title]
```

```
[-dept Department] [-company Company] [-mgr Manager]
[-hmdir HomeDirectory] [-hmdrv DriveLetter:] [-profile ProfilePath]
[-loscr ScriptPath] [-mustchpwd {yes | no}] [-canchpwd {yes | no}]
[-reversiblepwd {yes | no}] [-pwdneverexpires {yes | no}]
[-acctexpires NumberOfDays] [-disabled {yes | no}]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}]
[-q] [{-uc | -uco | -uci}] [-fnp FirstNamePhonetic]
[-lnp LastNamePhonetic] [-displayp DisplayNamePhonetic]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSGET COMPUTER

Displays the properties of a computer account using one of two syntaxes. The syntax for viewing the properties of multiple computers is

```
dsget computer ComputerDN ... [-dn] [-samid] [-sid] [-desc] [-loc]
[-disabled] [{-s Server | -d Domain}] [-u UserName]
[-p {Password | *}] [-c] [-q] [-l]  [{-uc | -uco | -uci}]
[-part PartitionDN [-qlimit] [-qused]]
```

The syntax for viewing the membership information of a single computer is

```
dsget computer ComputerDN [-memberof [-expand]]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}] [-c]
[-q] [-l] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSGET GROUP

Displays the properties of group accounts using one of two syntaxes. The syntax for viewing the properties of multiple groups is

```
dsget group GroupDN ... [-dn] [-samid] [-sid] [-desc] [-secgrp]
[-scope] [{-s Server | -d Domain}] [-u UserName] [-p {Password | *}]
[-c] [-q] [-l] [{-uc | -uco | -uci}] [-part PartitionDN [-qlimit]
[-qused]]
```

The syntax for viewing the group membership information for an individual group is

```
dsget group GroupDN [{-memberof | -members} [-expand]]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}]
[-c] [-q] [-l]  [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSGET SERVER

Displays the various properties of domain controllers using any of three syntaxes. The syntax for displaying the general properties of a specified domain controller is

```
dsget server ServerDN ... [-dn] [-desc] [-dnsname] [-site]
[-isgc] [{-s Server | -d Domain}] [-u UserName] [-p {Password | *}] [-c]
[-q] [-l] [{-uc | -uco | -uci}]
```

The syntax for displaying a list of the security principals who own the largest number of directory objects on the specified domain controller is

```
dsget server ServerDN [{-s Server | -d Domain}] [-u UserName]
[-p {Password | *}] [-c] [-q] [-l] [{-uc | -uco | -uci}]
[-topobjowner NumbertoDisplay]
```

The syntax for displaying the distinguished names of the directory partitions on the specified server is

```
dsget server ServerDN [{-s Server | -d Domain}] [-u UserName]
[-p {Password | *}] [-c] [-q] [-l] [{-uc | -uco | -uci}] [-part]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSGET USER

Displays the properties of user accounts using one of two syntaxes. The syntax for viewing the properties of multiple users is

```
dsget user UserDN ... [-dn] [-samid] [-sid] [-upn] [-fn] [-mi]
[-ln] [-display] [-fnp] [-lnp] [-displayp] [-effectivepso]
[-empid] [-desc] [-office] [-tel] [-email] [-hometel] [-pager]
[-mobile] [-fax] [-iptel] [-webpg] [-title] [-dept] [-company]
[-mgr] [-hmdir] [-hmdrv] [-profile] [-loscr] [-mustchpwd] [-canchpwd]
[-pwdneverexpires] [-disabled] [-acctexpires] [-reversiblepwd]
[{-uc | -uco | -uci}] [-part PartitionDN [-qlimit] [-qused]]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}] [-c] [-q]
[-l]
```

The syntax for viewing the group membership for users is

```
dsget user UserDN [-memberof [-expand]] [{-uc | -uco | -uci}]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}] [-c]
[-q] [-l]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSMGMT

Invokes a text-mode command interpreter so that you can manage directory services using a separate command prompt and commands that are internal to DSMGMT.

```
dsmgmt
```

DSMOD COMPUTER

Modifies attributes of one or more computer accounts in the directory.

```
dsmod computer ComputerDN ... [-desc Description] [-loc Location]
[-disabled {yes | no}] [-reset] [{-s Server | -d Domain}] [-u UserName]
[-p {Password | *}] [-c] [-q] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSMOD GROUP

Modifies attributes of one or more group accounts in the directory.

```
dsmod group GroupDN ... [-samid SAMName] [-desc Description]
[-secgrp {yes | no}] [-scope {l | g | u}]
[{-addmbr | -rmmbr | -chmbr} MemberDN ...] [{-s Server | -d Domain}]
[-u UserName] [-p {Password | *}] [-c] [-q] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSMOD SERVER

Modifies properties of a domain controller.

```
dsmod server ServerDN ... [-desc Description] [-isgc {yes | no}]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}] [-c]
[-q] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSMOD USER

Modifies attributes of one or more user accounts in the directory.

```
dsmod user UserDN ... [-upn UPN] [-fn FirstName] [-mi Initial]
[-ln LastName] [-display DisplayName] [-empid EmployeeID]
[-pwd {Password | *}] [-desc Description] [-office Office]
[-tel PhoneNumber] [-email EmailAddress] [-hometel HomePhoneNumber]
[-pager PagerNumber] [-mobile CellPhoneNumber] [-fax FaxNumber]
[-iptel IPPhoneNumber] [-webpg WebPage] [-title Title]
[-dept Department] [-company Company] [-mgr Manager]
[-hmdir HomeDirectory] [-hmdrv DriveLetter:] [-profile ProfilePath]
[-loscr ScriptPath] [-mustchpwd {yes | no}] [-canchpwd {yes | no}]
[-reversiblepwd {yes | no}] [-pwdneverexpires {yes | no}]
[-acctexpires NumberOfDays] [-disabled {yes | no}]
[{-s Server | -d Domain}] [-u UserName] [-p {Password | *}] [-c] [-q]
[{-uc | -uco | -uci}] [-fnp FirstNamePhonetic]
```

```
[-lnp LastNamePhonetic] [-displayp DisplayNamePhonetic]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSMOVE

Moves or renames Active Directory objects.

```
dsmove objectdn [-newname newname] [-newparent parentdn]
[{-s server | -d domain}] [-u username] [-p {password | *}] [-q]
[{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY COMPUTER

Searches for computer accounts matching criteria.

```
dsquery computer [{startnode | forestroot | domainroot}]
[-o {dn | rdn | samid}] [-scope {subtree | onelevel | base}] [-name
name]
[-desc description] [-samid samname] [-inactive numberofweeks]
[-stalepwd numberofdays] [-disabled] [{-s server | -d domain}]
[-u username] [-p {password | *}] [-q]   [-r] [-gc]
[-limit numberofobjects]
[{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY CONTACT

Searches for contacts matching criteria.

```
dsquery contact [{startnode | forestroot | domainroot}]
[-o {dn | rdn}] [-scope {subtree | onelevel | base}] [-name name]
[-desc description] [{-s server | -d domain}] [-u username]
```

```
[-p {password | *}] [-q] [-r] [-gc] [-limit numberofobjects]
[{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY GROUP

Searches for group accounts matching criteria.

```
dsquery group [{startnode | forestroot | domainroot}]
[-o {dn | rdn | samid}] [-scope {subtree | onelevel | base}]
[-name name] [-desc description] [-samid SAMName]
[{-s server | -d domain}] [-u username] [-p {password | *}] [-q]
[-r] [-gc] [-limit numberofobjects] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY PARTITION

Searches for Active Directory partitions matching criteria.

```
dsquery partition [-o {dn | rdn}] [-part filter] [-desc description] [{-s
server | -d domain}] [-u username] [-p {password | *}] [-q]
[-r] [-limit numberofobjects] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY QUOTA

Searches for disk quotas matching criteria.

```
dsquery quota {domainroot | objectdn} [-o {dn | rdn}] [-acct name]
[-qlimit filter] [-desc description] [{-s server | -d domain}]
[-u username] [-p {password | *}] [-q] [-r] [-limit numberofobjects]
[{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY SERVER

Searches for domain controllers matching criteria.

```
dsquery server [-o {dn | rdn}] [-forest] [-domain domainname]
[-site sitename] [-name name] [-desc description]
[-hasfsmo {schema | name | infr | pdc | rid}] [-isgc]
[-isreadonly] [{-s server | -d domain}] [-u username]
[-p {password | *}] [-q] [-r] [-gc] [-limit numberofobjects]
[{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY SITE

Searches for Active Directory sites matching criteria.

```
dsquery site [-o {dn | rdn}] [-name name] [-desc description]
[{-s server | -d domain}] [-u username] [-p {password | *}] [-q]
[-r] [-gc] [-limit numberofobjects] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY USER

Searches for user accounts matching criteria.

```
dsquery user [{startnode | forestroot | domainroot}]
[-o {dn | rdn | upn | samid}] [-scope {subtree | onelevel | base}]
[-name name] [-namep namephonetic] [-desc description] [-upn upn]
[-samid samname] [-inactive numberofweeks] [-stalepwd numberofdays]
[-disabled] [{-s server | -d domain}] [-u username]
[-p {password | *}] [-q] [-r] [-gc] [-limit numberofobjects]
[{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSQUERY *

Searches for any Active Directory objects matching criteria.

```
dsquery * [{startnode | forestroot | domainroot}]
[-scope {subtree | onelevel | base}] [-filter ldapfilter]
[-attr {attributelist | *}] [-attrsonly] [-l]
[{-s server | -d domain}] [-u username] [-p {password | *}] [-q]
[-r] [-gc]  [-limit numberofobjects] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

DSRM

Deletes Active Directory objects.

```
dsrm objectdn ... [-subtree [-exclude]] [-noprompt]
[{-s server | -d domain}] [-u username] [-p {password | *}] [-c]
[-q] [{-uc | -uco | -uci}]
```

This command is available in Windows 8.1 if Administration Tools Pack has been installed.

ECHO

Displays messages or turns command echoing on or off.

```
echo [on | off]
echo [message]
```

ENDLOCAL

Ends localization of environment changes in a batch file.

```
endlocal
```

ERASE

See DEL.

ESENTUTL

Manages Extensible Storage Engine (ESE) databases, including those used by Active Directory Domain Services (ADDS).

Syntax for defragmentation:

```
esentutl d databasename s [streamingfilename] /t [tempdbname]
/f [tempstreamingfilename] i p b [backupfilename] 8 /o
```

Syntax for recovery:

```
esentutl /r logfilebasename /l [logdirectory]
/s [systemfilesdirectory]
i t u [log] d [dbfiledirectory] n path1[:path2] 8 /o
```

Syntax for checking integrity:

```
esentutl /g databasename s [streamingfilename] t [tempdbname]
/f [tempstreamingfilename] i 8 /o
```

Syntax for performing a checksum:

```
esentutl /k filetocheck /s [streamingfilename]
/t [tempdbname] /p nn
/e i 8 /o
```

Syntax for repair:

```
esentutl /p databasename s [streamingfilename] t [tempdbname]
/f [reportprefix] i g /createstm 8 o
```

Syntax for dumping a file:

```
esentutl /m [h|k|l|m|s|u] filename /p pagenumber
/s [streamingfilename] /t tablename /v 8 o
```

Syntax for copying a file:

```
esentutl y sourcefile d destinationfile /o
```

EVENTCREATE

Creates custom events in the event logs.

```
eventcreate [/s computer [/u domain\user [/p password]]
[/l {application | system}] | [/so srcname]
t {success | error | warning | information} id eventid
/d description
```

EXIT

Exits the command interpreter.

```
exit [/b] [exitcode]
```

EXPAND

Uncompresses files.

```
expand [-r] source destination
expand -r source [destination]
expand -d source.cab [-f:files]
expand source.cab -f:files destination
```

FC

Compares files and displays differences.

```
fc [/a] [/c] [/l] [/lbn] [/n] [/t] [u] [w]
   [/nnnn][/offline][drive1:][path1]filename1
   [drive2:][path2]filename2
fc /b [drive1:][path1]filename1
   [drive2:][path2]filename2
```

FIND

Searches for a text string in files.

```
find [v] [c] [/n] [/i] [/offline] "string"
[[drive:][path]filename[ ...]]
```

FINDSTR

Searches for strings in files using regular expressions.

```
findstr [/b] [e] [l] [/r] [/s] [/i] [/x] [v] [n]
[/m] [o] [p] [/f:file] [/a:attr] [/c:string]
[/d:dir] [/g:file] [/offline] [strings]
[[drive:][path]filename[ ...]]
```

FOR

Runs a specified command for each file in a set of files.

Command-line FOR looping:

```
for %variable in (set) do command [parameters]
for /d %variable in (set) do command [parameters]
for /r [[drive:]path] %variable in (set) do command [parameters]
for /l %variable in (start,step,end) do command [parameters]
for /f ["options"] %variable in (set) do command [parameters]
```

Script FOR looping:

```
for %%variable in (set) do command [parameters]
for /d %%variable in (set) do command [parameters]
for /r [[drive:]path] %%variable in (set) do command [parameters]
for /l %%variable in (start,step,end) do command [parameters]
for /f ["options"] %%variable in (set) do command [parameters]
```

FORFILES

Selects one or more files and executes a command on each file.

```
forfiles [/p pathname] [/m searchmask] [/s] [/c command]
[/d [+ | -] {mm/dd/yyyy | dd}]
```

FORMAT

Formats a floppy disk or hard drive.

```
format drive: [/fs:file-system] [/v:label] [/q] [/a:size] [/c]
```

```
[/x] [/p:numzerofillpasses]
format drive: [/v:label] [/q] [/f:size | /t:tracks /n:sectors]
[/p:numzerofillpasses]
```

FTP

Transfers files.

```
ftp [-v] [-d] [-i] [-n] [-g] [-s:filename] [-a] [-A] [-x:sendbuffer]
[-r:recvbuffer] [-b:asyncbuffers] [-w:windowsize] [host]
```

The parameters for the FTP command are case-sensitive. You must enter them in the letter case shown.

FTYPE

Displays or modifies file types used in file extension associations.

```
ftype [fileType[=[command]]]
```

GET-EVENTLOG

A Windows PowerShell command for displaying information about event logs or entries stored in event logs. (Only works with classic event logs.)

```
get-eventlog -list
get-eventlog [-logname] logname [-newest nn]
```

GET-PROCESS

A Windows PowerShell command for displaying information about running processes.

```
get-process -id [id1,id2,...]
get-process -inputobject processname1, processname2,... [process ...]
get-process [-name] [processname1, processname2,...]
```

GET-SERVICE

A Windows PowerShell command for displaying information about configured services.

```
get-service [-displayname [servicename1, servicename2,...]]
 -include [servicename1, servicename2,...]
 -exclude [servicename1, servicename2,...]
get-service [-name] [servicename1, servicename2,...]
 -include [servicename1, servicename2,...]
 -exclude [servicename1, servicename2,...]
get-service [-inputobject servicename1, servicename2,...]
 [-include [servicename1, servicename2,...]
 [-exclude [servicename1, servicename2,...]
```

GETMAC

Displays network adapter information.

```
getmac [/s computer [/u [domain]\user [/p [pwd]]]]
[/fo {table|list|csv}] [/nh] [/v]
```

GOTO

Directs the Windows command interpreter to a labeled line in a script.

```
goto :label
goto :EOF
```

GPUPDATE

Forces a background refresh of group policy.

```
gpupdate [/target:{computer | user}] [/force] [/wait:<value>]
[/logoff] [/boot] [/sync]
```

HOSTNAME

Prints the computer's name.

```
hostname
```

ICACLS

Displays or modifies a file’s access control list (ACL).

Syntax for storing ACLs for all matching names into an ACL file:

```
icacls name /save aclfile [/t] [/c] [/l] [/q]
```

Syntax for restoring stored ACLs to files in a directory:

```
icacls directory [/substitute sidold sidnew [...]] /restore aclfile
[/c] [/l] [/q]
```

Syntax for changing the owner for all matching names:

```
icacls name /setowner user [/t] [/c] [/l] [/q]
```

Syntax for finding all matching names with a particular SID applied:

```
icacls name /findsid sid [/t] [/c] [/l] [/q]
```

Syntax for granting permissions:

```
icacls name [/grant[:r] sid:perm[...]]
```

Syntax for denying permissions:

```
icacls name [/deny sid:perm [...]]
```

Syntax for removing permissions:

```
icacls name [/remove[:g|:d]] sid[...]] [/t] [/c] [/l] [/q]
```

Syntax for resetting ACLs to inherited values:

```
icacls name reset [t] [/c] [/l] [/q]
```

Syntax for setting an integrity level:

```
icacls name [/setintegritylevel level:policy[...]]
```

Syntax for verifying ACLs:

```
icacls name verify [t] [/c] [/l] [/q]
```

IF

Performs conditional processing in batch programs.

```
if [not] errorlevel number command
if [not] [/i] string1==string2 command
if [not] exist filename command
if [/i] string1 compare-op string2 command
if cmdextversion number command
if defined variable command
```

IPCONFIG

Displays TCP/IP configuration.

```
ipconfig [/allcompartments] {/all}
ipconfig [/release [adapter] | /renew [adapter]
         | /release6 [adapter] | /renew6 [adapter]]
ipconfig /flushdns | displaydns | registerdns
ipconfig /showclassid adapter
ipconfig /setclassid adapter [classidtoset]
ipconfig /showclassid6 adapter
ipconfig /setclassid6 adapter [classidtoset]
```

LABEL

Creates, changes, or deletes the volume label of a disk.

```
label [drive:][label]
label [/mp] [volume] [label]
```

MD

Creates a directory or subdirectory.

```
mkdir [drive:]path
md [drive:]path
```

MKDIR

See MD.

MORE

Displays output one screen at a time.

```
more [e [c] [/p] [/s] [/tn] [+n]] < [drive:][path]filename
more e [c] [/p] [/s] [/tn] [+n] [files]
command–name | more [e [c] [/p] [/s] [/tn] [+n]]
```

MOUNTVOL

Manages volume mount point.

```
mountvol [drive:]path volumeName
mountvol [drive:]path {d | l | /p}
mountvol [/r | n | e]
```

MOVE

Moves files from one directory to another directory on the same drive.

```
move [/y] [/-y] source target
```

NBTSTAT

Displays status of NETBIOS.

```
nbtstat [-a remotename] [-A ipaddress] [-c] [-n] [-r] [-R] [-RR]
[-s] [-S] [interval]
```

NOTE This command uses case-sensitive switches.

NET ACCOUNTS

Manage user account and password policies.

```
net accounts [/forcelogoff:{minutes | no}]
```

```
 [/minpwlen:length]
 [/maxpwage:{days | unlimited}]
 [/minpwage:days]
 [/uniquepw:number] [/domain]
```

NET COMPUTER

Adds or remove computers from a domain.

```
net computer \\computername {/add | /del}
```

NET CONFIG SERVER

Displays or modifies configuration of server service.

```
net config server [/autodisconnect:time]
   [/srvcomment:"text"] [/hidden:{yes | no}]
```

NET CONFIG WORKSTATION

Displays or modifies configuration of workstation service.

```
net config workstation [/charcount:bytes]
[/chartime:msec]
[/charwait:sec]
```

NET CONTINUE

Resumes a paused service.

```
net continue service
```

NET FILE

Displays or manages open files on a server.

```
net file [id [/close]]
```

NET GROUP

Displays or manages global groups.

```
net group [groupname [/comment:"text"]]
  [/domain]
net group groupname {add [comment:"text"]
  | delete} [domain]
net group groupname username [...]
  {/add | delete} [domain]
```

NET LOCALGROUP

Displays local group accounts.

```
net localgroup [GroupName [/comment:"Text"]] [/domain]
```

Creates a local group account.

```
net localgroup GroupName {add [comment:"Text"]} [/domain]
```

Modifies local group accounts.

```
net localgroup [GroupName Name [ ...] add [domain]
```

Deletes a local group account.

```
net localgroup GroupName delete  [domain]
```

NET PAUSE

Suspends a service.

```
net pause service
```

NET PRINT

Displays or manages print jobs and shared queues.

```
net print \\computername\sharename
net print [\\computername] job# [/hold | release | delete]
```

NET SESSION

Lists or disconnects sessions.

```
net session [\\computername] [/delete] [/list]
```

NET SHARE

Displays or manages shared printers and directories.

```
net share [sharename]
net share sharename[=drive:path] [/users:number | /unlimited]
  [/remark:"text"] [/cache:flag]
net share {sharename | devicename | drive:path} /delete
```

NET START

Lists or starts network services.

```
net start [service]
```

NET STATISTICS

Displays workstation and server statistics.

```
net statistics [workstation | server]
```

NET STOP

Stops services.

```
net stop service
```

NET TIME

Displays or synchronizes network time.

```
net time [\\computername | /domain[:domainname] |
 /rtsdomain[:domainname]] [/set]
net time [\\computername] /querysntp
net time [\\computername] /setsntp[:serverlist]
```

NET USE

Displays or manages remote connections.

```
net use [devicename | *] [\\computername\sharename[\volume]
[password | *]] [/user:[domainname\]username]
[/user:[username@domainname]] [[/delete] | [/persistent:{yes | no}]]
[/smartcard] [/savecred]
net use [devicename | *] [password | *]] [/home]
net use [/persistent:{yes | no}]
```

NET USER

Creates local user accounts.

```
net user UserName [Password | *] add [active:{no | yes}]
[/comment:"DescriptionText"] [/countrycode:NNN]
[/expires: {{MM/DD/YYYY | DD/MM/YYYY | mmm,dd,YYYY} | never}]
[/fullname:"Name"] [/homedir:Path] [/passwordchg:{yes | no}]
[/passwordreq:{yes | no}] [/profilepath:[Path]] [/scriptpath:Path]
[/times:{Day[-Day][,Day[-Day]] ,Time[-Time][,Time[-Time]]
[;...] | all}] [/usercomment:"Text"]
[/workstations:{ComputerName[,...] | *}] [/domain]
```

Modifies local user accounts.

```
net user [UserName [Password | *] [/active:{no | yes}]
[/comment:"DescriptionText"] [/countrycode:NNN]
[/expires:{{MM/DD/YYYY | DD/MM/YYYY | mmm,dd,YYYY} | never}]
[/fullname:"Name"] [/homedir:Path] [/passwordchg:{yes | no}]
[/passwordreq:{yes | no}] [/profilepath:[Path]] [/scriptpath:Path]
[/times:{Day[-Day][,Day[-Day]] ,Time[-Time][,Time[-Time]]
[;...] | all}] [/usercomment:"Text"]
[/workstations:{ComputerName[,...] | *}]] [/domain]
```

Deletes local user accounts.

```
net user UserName [/delete] [/domain]
```

NET VIEW

Displays network resources or computers.

```
net view [\\computername [/cache] | [/all] |
 /domain[:domainname]]
```

NETDOM ADD

Adds a workstation or server account to the domain.

```
netdom add computer [/domain:domain] [/userd:user]
[/passwordd:[password | *]]
[/server:server] [/ou:oupath] [/dc] [/securepasswordprompt]
```

NETDOM COMPUTERNAME

Manages the primary and alternate names for a computer. This command can safely rename a domain controller or a server.

```
netdom computername computer [/usero:user]
[/passwordo:[password | *]]
[/userd:user] [/passwordd:[password | *]] [/securepasswordprompt]
/add:newalternatednsname | /remove:alternatednsname |
    /makeprimary:computerdnsname |
/enumerate[:{alternatenames | primaryname | allnames}] | /verify
```

NETDOM JOIN

Joins a workstation or member server to the domain.

```
netdom join computer /domain:domain [/ou:oupath] [/userd:user]
[/passwordd:[password | *]]
[/usero:user] [/passwordo:[password | *]]
[/reboot[:timeinseconds]]
```

```
[/securepasswordprompt]
```

NETDOM MOVE

Moves a workstation or member server to a new domain.

```
netdom move computer /domain:domain [/ou:oupath]
[/userd:user] [/passwordd:[password | *]]
[/usero:user] [/passwordo:[password | *]]
[/userf:user] [/passwordf:[password | *]]
[/reboot[:timeinseconds]]
[/securepasswordprompt]
```

NETDOM QUERY

Queries a domain for information.

```
netdom query [/domain:domain] [/server:server]
[/userd:user] [/passwordd:[password | *]]
[/verify] [/reset] [/direct] [/securepasswordprompt]
{workstation | server | dc | ou | pdc | fsmo | trust}
```

NETDOM REMOVE

Removes a workstation or server from the domain.

```
netdom remove computer [/domain:domain] [/userd:user]
[/passwordd:[password | *]]
[/usero:user] [/passwordo:[password | *]]
[/reboot[:timeinseconds]] [/force]
[/securepasswordprompt]
```

NETDOM RENAMECOMPUTER

Renames a computer. If the computer is joined to a domain, the computer object in the domain is also renamed. You should not use this command to rename a domain controller.

```
netdom renamecomputer computer /newname:newname
```

```
[/userd:user [/passwordd:[password | *]]]
[/usero:user [/passwordo:[password | *]]]
[/force] [/reboot[:timeinseconds]]
[/securepasswordprompt]
```

NETDOM RESET

Resets the secure connection between a workstation and a domain controller.

```
netdom reset computer [/domain:domain] [/server:server]
[/usero:user] [/passwordo:[password | *]] [/securepasswordprompt]
```

NETDOM RESETPWD

Resets the machine account password for the domain controller on which this command is run.

```
netdom resetpwd /server:domaincontroller /userd:user
/passwordd:[password | *]
[/securepasswordprompt]
```

NETDOM TRUST

Manages or verifies the trust relationship between domains.

```
netdom trust trustingdomainname /domain:trusteddomainname [/userd:user]
[/passwordd:[password | *]] [/usero:user] [/passwordo:[password | *]]
[/verify] [/reset] [/passwordt:newrealmtrustpassword]
[/add] [/remove] [/twoway] [/realm] [/kerberos]
[/transitive[:{yes | no}]]
[/oneside:{trusted | trusting}] [/force] [/quarantine[:{yes | no}]]
[/namesuffixes:trustname [/togglesuffix:#]]
[/enablesidhistory[:{yes | no}]]
[/foresttransitive[:{yes | no}]]
[/crossorganization[:{yes | no}]]
[/addtln:toplevelname]
[/addtlnex:toplevelnameexclusion]
[/removetln:toplevelname]
[/removetlnex:toplevelnameexclusion]
[/securepasswordprompt]
```

```
[/enabletgtdelegation[:yes | no>]]
```

NETDOM VERIFY

Verifies the secure connection between a workstation and a domain controller.

```
netdom verify computer [/domain:domain] [/usero:user]
   [/passwordo:[password | *]] [/securepasswordprompt]
```

NETSH

Invokes a separate command prompt that allows you to manage the configuration of various network services on local and remote computers.

```
netsh
```

NETSTAT

Displays status of network connections.

```
netstat [-a] [-b] [-e] [-f] [-n] [-o] [-p protocol] [-r] [-s] [-t] [-x]
[interval]
```

NSLOOKUP

Shows the status of DNS.

```
nslookup [-option] [computer]
nslookup [-option] [computer server]
```

PATH

Displays or sets a search path for executable files in the current command window.

```
path [[drive:]path[;...][;%PATH%]
path ;
```

PATHPING

Traces routes and provides packet loss information.

```
pathping [-n] [-h maxhops] [-g hostlist]
 [-i address] [-p period]
 [-q numqueries] [-w timeout]
 targetname [-4] [-6]
```

PAUSE

Suspends processing of a script and waits for keyboard input.

```
pause
```

PING

Determines whether a network connection can be established.

```
ping [-t] [-a] [-n count] [-l size] [-f]
 [-i TTL] [-v TOS] [-r count] [-s count]
 [[-j hostlist] | [-k hostlist]]
 [-w timeout] [-R} [-S sourceaddress]
 [-4] [-6] destinationlist
```

> *NOTE* This command uses case-sensitive switches.

POPD

Changes to the directory stored by PUSHD.

```
popd
```

PRINT

Prints a text file.

```
print [/d:device]
[[drive:][path]filename[...]]
```

PROMPT

Changes the Windows command prompt.

```
prompt [text]
```

PUSHD

Saves the current directory then changes to a new directory.

```
pushd [path | ..]
```

RD

Removes a directory.

```
rmdir [/s] [/q] [drive:]path
rd [/s] [/q] [drive:]path
```

RECOVER

Recovers readable information from a bad or defective disk.

```
recover [drive:][path]filename
```

REG ADD

Adds a new subkey or entry to the registry.

```
reg add keyname [/v valuename | ve] [t datatype] [/d data] [/f]
[/s separator] [/reg32 | /reg64]
```

REG COMPARE

Compares registry subkeys or entries.

```
reg compare keyname1 keyname2 [/v valuename | ve] [s]
[/outputoption]
```

REG COPY

Copies a registry entry to a specified key path on a local or remote system.

```
reg copy keyname1 keyname2 [/s] [/f]
```

REG DELETE

Deletes a subkey or entries from the registry.

```
reg delete keyname [/v valuename | ve | va] [/f]
```

REG QUERY

Lists the entries under a key and the names of subkeys (if any).

```
reg query keyname [/v valuename | ve] [s]
[/f data [/k] [/d] [/c] [/e]] [/t type] [/z] [/se separator]
```

REG RESTORE

Writes saved subkeys and entries back to the registry.

```
reg restore keyname "filename"
```

REG SAVE

Saves a copy of specified subkeys, entries, and values to a file.

```
reg save keyname "filename" [/y]
```

REGSVR32

Registers and unregisters DLLs.

```
regsvr32 [u] [s] [/n] [/i[:cmdline]] dllname
```

REM

Adds comments to scripts.

```
rem [comment]
```

REN

Renames a file.

```
rename [drive:][path]filename1 filename2
ren [drive:][path]filename1 filename2
```

RMDIR

See RD.

ROUTE

Manages network routing tables.

```
route [-f] [-p] [-4|-6] command [destination]
[mask netmask] [gateway] [metric metric] [if interface]
```

RUNAS

Runs program with specific user permissions.

Syntax for running with a specified user's credentials:

```
runas [/noprofile | profile] [env] [/netonly] [/savecred]
        /user:account program
```

Syntax for running with credential from a smart card:

```
runas [/noprofile | profile] [env] [/netonly] [/savecred]
smartcard [user:account] program
```

Syntax for showing available trust levels:

```
runas /showtrustlevels
```

Syntax for running a program at a specified trust level:

```
runas /trustlevel:trustlevel program
```

SC CONFIG

Configures service startup and logon accounts.

```
sc [\\ServerName] config ServiceName
  [type= {own | share|{interact type = {own | share}} | kernel |
filesys |rec | adapt}]
  [start= {boot | system | auto | demand | disabled | delayed-auto}]
  [error= {normal | severe | critical | ignore}]
  [binPath= BinaryPathName]
  [group= LoadOrderGroup]
  [tag= {yes | no}]
  [depend= Dependencies]
  [obj= {AccountName | ObjectName}]
  [DisplayName= displayname]
  [password= password]
```

SC CONTINUE

Resumes a paused service.

```
sc [\\ServerName] continue ServiceName
```

SC FAILURE

Views the actions that will be taken if a service fails.

```
sc [\\ServerName] failure ServiceName [reset= ErrorFreePeriod]
   [reboot= BroadcastMessage] [command= CommandLine]
   [actions= FailureActionsAndDelayTime]
```

SC PAUSE

Pauses a service.

```
sc [\\ServerName] pause ServiceName
```

SC QC

Displays configuration information for a named service.

```
sc [\\ServerName] qc ServiceName [BufferSize]
```

SC QFAILURE

Sets the action to take upon failure of a service.

```
sc [\\ServerName] qfailure ServiceName [BufferSize]
```

SC QUERY

Displays the list of services configured on the computer.

```
sc [\\ServerName] query ServiceName
  [type= {driver | service | all}]
  [type= {own|share|interact|kernel|filesys|rec|adapt}]
  [state= {active | inactive | all}] [bufsize= BufferSize]
  [ri= ResumeIndex]
  [group= GroupName]
```

SC START

Starts a service.

```
sc [\\ServerName] start ServiceName [ServicesArgs]
```

SC STOP

Stops a service

Stops a service.

```
sc [\\ServerName] stop ServiceName
```

SCHTASKS /CHANGE

Changes the properties of existing tasks.

```
schtasks /change /tn taskname [/s system [/u [domain\]user
[/p [password]]]] {[/ru [domain\]user]
[/rp password]  [/tr tasktorun]} [/st starttime] [/ri runintrrval]
[{/et endtime | /du duration} [/k]] [/sd startdate] [/ed enddate]
[enable | disable] [/it] [/z]
```

SCHTASKS /CREATE

Creates scheduled tasks.

```
schtasks /create [/s system [/u [domain\]user [/p [password]]]]
[/ru [domain\]username [rp password]] /tn taskname /tr tasktorun
/sc scheduletype [/mo modifier] [/d day] [/i idletime]
[/st starttime] [/m month [, month [...]]] [/sd startdate]
[/ed enddate] [/ri runintrrval] [{/et endtime | /du duration} [/k]]
[/it | np] [z] [/f] [/xml xmlfile]
```

SCHTASKS /DELETE

Removes scheduled tasks that are no longer wanted.

```
schtasks /delete /tn {TaskName | *} [/f] [/s computer
[/u [domain\]user [/p [password]]]]
```

SCHTASKS /END

Stops a running task.

```
schtasks /end /tn taskname [/s computer [/u [domain\]user
[/p [password]]]]
```

SCHTASKS /QUERY

Displays scheduled tasks on the local or named computer.

```
schtasks /query [/s computer [/u [domain\]user [/p [password]]]]
[/fo {table | list | csv} | xml] [nh] [v] [tn {TaskName][/HRESULT]
```

SCHTASKS /RUN

Starts a scheduled task.

```
schtasks /run /tn taskname [/s computer [/u [domain\]user
[/p [password]]]] [/I] [/HRESULT]
```

SET

Displays or modifies Windows environment variables. Also used to evaluate numeric expressions at the command line.

```
set [variable=[string]]
set /a expression
set /p variable=[promptstring]
```

SET-SERVICE

A Windows PowerShell command for modifying the configuration of system services.

```
set-service [-name] servicename [-displayname displayname]
 [-description description]
 [-startuptype {Automatic|Manual|Disabled}] [-whatif] [-config]
[parameters]
```

SETLOCAL

Begins localization of environment changes in a batch file.

```
setlocal
setlocal {enableext | disableext}
```

SFC

Scans and verifies protected system files.

```
sfc [/scannow] [/verifyonly] [/scanfile=file] [/verifyfile=file]
[/offwindir=offlinewindowsdirectory /offbootdir=offlinebootdirectory]
```

SHIFT

Shifts the position of replaceable parameters in scripts.

```
shift [/n]
```

SHUTDOWN

Shuts down or restarts a computer.

```
shutdown [/i | l | s | r | g | a | p | h | e | o] [hybrid] [/f] [/m
\\computerName][/t nn][/d [p|u:]n1:n2
[/c "comment"]]
```

SORT

Sorts input.

```
[command |] sort [/r] [/+n] [/m kb] [/l locale] [/rec recordbytes]
[drive1:][path1]filename1] [/t [drive2:][path2]]
[/o [drive3:][path3]filename3]
```

NOTE In this instance, the symbol "|" is the pipe symbol.

START

Starts a new command-shell window to run a specified program or command.

```
start ["title"] [/d path] [/i] [/min] [/max] [/separate | /shared]
[/wait] [/b] [/low | belownormal | normal | /abovenormal
| /high | realtime] [affinity hh] [command/program] [parameters]
```

STOP-PROCESS

A Windows PowerShell command for stopping one or more running processes.

```
stop-process –id [id1,id2,...] [-confirm] [-passthru] [-whatif]
             [parameters]
stop-process –inputobject processname1, processname2,... [-passthru]
             [-whatif] [-config] [parameters]
stop-process -name processname1, processname2,... [-confirm]
             [-passthru] [-whatif] [parameters]
```

STOP-SERVICE

A Windows PowerShell command for stopping one or more running services.

```
stop-service [–displayname [servicename1, servicename2,...]]
 -include [servicename1, servicename2,...]
 -exclude [servicename1, servicename2,...]
stop-service [-name] [servicename1, servicename2,...]
 -include [servicename1, servicename2,...]
 -exclude [servicename1, servicename2,...]
```

> *NOTE* Windows PowerShell also has commands for starting (start-service), restarting (restart-service), suspending (suspend-service), and resuming (resume-service) services. These commands have the same syntax as stop-service.

SUBST

Maps a path to a drive letter.

```
subst [drive1: [drive2:]path]
subst drive1: /d
```

SYSTEMINFO

Displays detailed configuration information.

```
systeminfo [/s computer [/u [domain\]user [/p [pwd]]]]
[/fo {table|list|csv}] [/nh]
```

TAKEOWN

Allows an administrator to take ownership of one or more files.

```
takeown [/s computer [/u [domain\]user [/p [pwd]]]] /f filename
[/a] [/r [/d prompt]]
```

TASKKILL

Stops running processes by name or process ID.

```
taskkill [/s computer] [/u [domain\]user [/p pwd]]] {[/fi filter1
[/fi filter2 [ ... ]]] [/pid ID|/im imgName]} [/f][/t]
```

TASKLIST

Lists all running processes by name and process ID.

```
tasklist [/s computer [/u [domain\]user [/p [password]]]]
[{/m module | svc | v}] [/fo {table | list | csv}] [/nh]
[/fi filtername [/fi filtername2 [ ... ]]]
```

TIME

Displays or sets the system time.

```
time [time | /T]
```

TIMEOUT

Sets a timeout period or waits for key press in batch script.

```
TIMEOUT /t timeout [/nobreak]
```

TITLE

Sets the title for the command-shell window.

```
title [string]
```

TRACERPT

Generates trace reports from trace logs.

```
tracerpt {[-l] logfile1 logfile2 ... | [-o outputfile] |
-rt sessionname1 sessionname2 ...} [-of <CSV | EVTX | XML>]
[-lr] [-summary summaryreportfile] [-report reportfilename]
[-f <XML | HTML>] [-df schemafile] [-int dumpeventfile] [-rts]
[-tmf tracedefinitionfile] [-tp tracefilepath] [-gmt] [-i imagepath]
[-pdb symbolpath] [-rl n] [-export schemaexportfile]
[-config configfile] [-y]
```

TRACERT

Displays the path between computers.

```
tracert [-d] [-h maximumhops] [-j hostlist] [-w timeout]
[-r] [-s sourceaddrress] [-4] [-6] targetname
```

TYPE

Displays the contents of a text file.

```
type [drive:][path]filename
```

TYPEPERF

Displays or logs performance data for specified counts.

```
typeperf {{counter1 counter2 ...} | -cf counterfile} [-sc numsamples]
     [-si {[[hh:]mm:]ss] [-o logfile] [ -f {CSV | TSV | BIN | SQL}]
     [-s server] [-y]
typeperf {-q object | -qx object} [-sc numsamples ]
     [-si {[[hh:]mm:]ss] [-o logfile] [ -f {CSV | TSV | BIN | SQL}]
     [-s server] [-y]
```

VER

Displays the Windows version.

```
ver
```

VERIFY

Tells Windows whether to verify that your files are written correctly to a disk.

```
verify [on | off]
```

VOL

Displays a disk volume label and serial number.

```
vol [drive:]
```

WAITFOR

Specifies that a computer should wait for a particular signal before continuing.

Send signal syntax:

```
waitfor [/s computer [/u [domain\]user [/p [pwd]]]] /si signal
```

Wait signal syntax:

```
waitfor [/t timeout] signal
```

WBADMIN

Performs or schedules backup and recovery operations. This command applies only to Windows 2012, Windows Server 2012 R2 and Windows 8.1 Business, Enterprise, and Ultimate Editions.

Syntax for enabling backups:

```
wbadmin enable backup
    [-addtarget:{BackupTargetDisk}]
    [-removetarget:{BackupTargetDisk}]
    [-schedule:TimeToRunBackup] [-include:VolumesToInclude]
    [-allCritical] [-quiet]
```

Syntax for disabling backups:

```
wbadmin disable backup [-quiet]
```

Syntax for starting backups:

```
wbadmin start backup
[-backupTarget:{TargetVolume | TargetNetworkShare}]
[-include:VolumesToInclude] [-allCritical]
[-noVerify] [-user:UserName] [-password:Password]
[-noInheritAcl] [-vssFull] [-quiet]
```

Syntax for stopping the current backup job:

```
wbadmin stop job [-quiet]
```

Syntax for listing available disks:

```
wbadmin get disks
```

Syntax for getting the status of current backup job:

```
wbadmin get status
```

Syntax for getting a list of available backups:

```
wbadmin get versions
[-backupTarget:{VolumeName | NetworkSharePath}]
[-machine:BackupMachineName]
```

Syntax for starting a system state backup:

```
wbadmin start systemstatebackup
-backupTarget:{VolumeName} [-quiet]
```

Syntax for starting a system state recovery:

Syntax for starting a system state recovery:

```
wbadmin start systemstaterecovery
-version:VersionIdentifier -showsummary
[-backupTarget:{VolumeName | NetworkSharePath}]
[-machine:BackupMachineName]
[-recoveryTarget:TargetPathForRecovery]
[-authsysvol] [-quiet]
```

Syntax for deleting a system state backup:

```
wbadmin delete systemstatebackup
-keepVersions: NumberCopiesToKeep | -version VersionID |
-deleteOldest
[-backupTarget:{VolumeName}] [-machine:BackupMachineName]
[-quiet]
```

WHERE

Displays a list of files that match a search pattern.

```
where [/r dir] [/q] [/f] [/t] pattern
where [/q] [/f] [/t] $env:pattern
where [/q] [/f] [/t] path:pattern
```

WHOAMI

Displays log on and security information for the current user.

```
whoami [/upn | fqdn | logonid]
whoami {[/user] [/groups] [/claims] [/priv]} [/fo {table|list|csv}]
[/nh]
whoami all [fo {table|list|csv}] [/nh]
```

About the Author

William Stanek (http://www.williamstanek.com/) has more than 20 years of hands-on experience with advanced programming and development. He is a leading technology expert, an award-winning author, and a pretty-darn-good instructional trainer. Over the years, his practical advice has helped millions of programmers, developers, and network engineers all over the world. His current and books include *Windows 8.1 Administration Pocket Consultants*, *Windows Server 2012 R2 Pocket Consultants* and *Windows Server 2012 R2 Inside Outs*.

William has been involved in the commercial Internet community since 1991. His core business and technology experience comes from more than 11 years of military service. He has substantial experience in developing server technology, encryption, and Internet solutions. He has written many technical white papers and training courses on a wide variety of topics. He frequently serves as a subject matter expert and consultant.

William has an MS with distinction in information systems and a BS in

William has an MS with distinction in information systems and a BS in computer science, magna cum laude. He is proud to have served in the Persian Gulf War as a combat crewmember on an electronic warfare aircraft. He flew on numerous combat missions into Iraq and was awarded nine medals for his wartime service, including one of the United States of America’s highest flying honors, the Air Force Distinguished Flying Cross. Currently, he resides in the Pacific Northwest with his wife and children.

William recently rediscovered his love of the great outdoors. When he’s not writing, he can be found hiking, biking, backpacking, traveling, or trekking in search of adventure with his family!

Find William on Twitter at www.twitter.com/WilliamStanek and on Facebook at www.facebook.com/William.Stanek.Author.