THE ADDISON-WESLEY MICROSOFT TECHNOLOGY SERIES

ESSENTIAL POWERSHELL

HOLGER SCHWICHTENBERG

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: www.informit.com/aw

Library of Congress Cataloging-in-Publication Data

Schwichtenberg, Holger.
Essential PowerShell / Holger Schwichtenberg.
p. cm.
ISBN 978-0-672-32966-1
1. Windows PowerShell (Computer programming language) 2. Command languages (Computer science) 3. Scripting languages (Computer science) 4. Systems programming (Computer science) 5. Microsoft Windows (Computer file) I. Title.
QA76.73.W56S39 2008
005.4'2—dc22

2008020010

ISBN-13: 978-0-672-32966-1
ISBN-10: 0-672-2966-2

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing June 2008

This Book Is Safari Enabled

The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- Go to http://www.informit.com/onlineedition
- Complete the brief registration form
- Enter the coupon code LJRM-BWZP-1HUA-KCPF-YA2S

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please e-mail customer-service@safaribooksonline.com.

Editor-in-Chief
Karen Gettman

Executive Editor
Neil Rowe

Development Editor
Mark Renfrow

Managing Editor
Kristy Hart

Project Editor
Betsy Harris

Copy Editor
Keith Cline

Indexer
Publishing Works, Inc.

Proofreader
Paula Lowell

Technical Editor
Tony Bradley

Publishing Coordinator
Cindy Teeters

Cover Designer
Gary Adair

Compositor
Nonie Ratcliff

PREFACE

Windows PowerShell is one of the most amazing products Microsoft has released in recent years, because it brings console-based system administration and scripting to the next level of abstraction. PowerShell is an excellent replacement for classic Windows shell commands and for Windows Script Host (WSH). PowerShell copies a lot of good features from UNIX shells and combines them with the power of the .NET Framework. In contrast to WSH, PowerShell enables consistent, straightforward, command-line system administration that does not require much software development knowledge.

Unfortunately, in the first version of PowerShell, the number of high-level commands is limited. For many tasks, lower-level concepts are required, especially the .NET Framework and Windows Management Instrumentation (WMI).

What Does This Book Cover?

This book covers the standard PowerShell commandlets, additional free commandlets (for example, PowerShell Community Extensions), and the direct use of classes from the .NET Framework, the Component Object Model (COM), WMI, and the Active Directory Service Interface (ADSI).

Because PowerShell is an extensive topic, this book cannot provide an exhaustive reference of all PowerShell commands and solutions for all possible administrative tasks. However, you will find a concise introduction to the most common command and scenarios. For more detailed information about PowerShell, refer to the Microsoft documentation for PowerShell, WMI, ADSI, and the .NET Framework (approximately 100,000 pages) as an additional source.

Who Should Read This Book?

The primary target audience comprises Windows administrators seeking a method of automated system administration that is more powerful than the classic Windows Shell but less complex than WSH and the associated COM components. After reading this book, administrators will be able to use PowerShell as their day-to-day command-line interface for all administrative tasks.

As a prerequisite, aside a good knowledge of the Windows operation system, you should have a basic understanding of object-oriented programming languages. Basic concepts of object orientation such as classes, objects, attributes, and methods are not explained in this book.

How This Book Is Structured

This book is organized into 24 chapters, some of which, based on your previous experience and knowledge of certain concepts, you might find easier to understand than others. The 24 chapters are split into two parts:

- **Part I: Getting Started with PowerShell.** Part I introduces the PowerShell architecture, all basic concepts (such as pipelining and navigation), the PowerShell Script Language, and the tools you should know.
- **Part II: Windows PowerShell in Action.** Part II covers PowerShell script solutions for day-to-day administrative tasks related to Windows services and Windows application, such as file system, processes, event logs, registry, networking, printers, documents, databases, Active Directory, and software installation. Each chapter contains dozens of self-contained examples.

The appendixes contain a list of all commandlets from PowerShell 1.0, the PowerShell Community Extensions 1.1.1, and the www.IT-Visions.de PowerShell Extensions 2.0. You will also find a short preview of the next version of Windows PowerShell (Version 2.0).

Throughout the text, you will find codes that match up to codes in Appendix C, "Bibliography." These codes are encased in brackets (for example, [MS01]). The appendix lists the code, the correlating subject, and

a link that will provide you with more information.

Occasionally, when a line of code is too long to fit on one line in the printed text, a code-continuation character has been used to show that the line continues. For example

```
"{0} can be reached at {1}.
➥This information is dated: {2:D}." -f $a, $b, $c
```

This Book's Website

Many of the scripts are available for download from its website, www.Windows-Scripting.com. This website also contains errata for this book and the option to offer feedback to the author.

CHAPTER 1

FIRST STEPS WITH WINDOWS POWERSHELL

In this chapter:

This chapter introduces Windows PowerShell and helps you set up your environment. In addition, the chapter provides a few easy examples that demonstrate how to use PowerShell.

What Is Windows PowerShell?

Windows PowerShell (WPS) is a new .NET-based environment for console-based system administration and scripting on Windows platforms. It includes the following key features:

- A set of commands called *commandlets*
- Access to all system and application objects provided by Component Object Model (COM) libraries, the .NET Framework, and Windows Management Instrumentation (WMI)
- Robust interaction between commandlets through pipelining based on typed objects

- A common navigation paradigm for different hierarchical or flat information stores (for example, file system, registry, certificates, Active Directory, and environment variables)
- An easy-to-learn, but powerful scripting language with weak and strong variable typing
- A security model that prevents the execution of unwanted scripts
- Tracing and debugging capabilities
- The ability to host WPS in any application

This book includes syntax and examples for these features, except the last one, which is an advanced topic that requires in-depth knowledge of a .NET language such as C#, C++/CLI, or Visual Basic .NET.

A Little Bit of History

The DOS-like command-line window survived many Windows versions in almost unchanged form. With WPS, Microsoft now provides a successor that does not just compete with UNIX shells, it surpasses them in robustness and elegance. WPS could be called an adaptation of the concept of UNIX shells on Windows using the .NET Framework, with connections to WMI.

Active Scripting with Windows Script Host (WSH, pronounced "wish") is much too complex for many administrators because it presupposes much knowledge about object-oriented programming and COM. The many exceptions and inconsistencies in COM make WSH and the associated component libraries hard to learn.

Even during the development of Windows Server 2003, Microsoft admitted that it had asked UNIX administrators how they administer their operating system. The short-term result was a large number of additional command-line tools included in Windows Server 2003. However, the long-term goal was to replace the DOS-like command-line window of Windows with a new, much more powerful shell.

Upon the release of the Microsoft .NET Framework in 2002, many people were expecting a "WSH.NET." However, Microsoft stopped the development of a new WSH for the .NET Framework because it foresaw that using .NET-based programming languages such as C# and Visual Basic .NET would require administrators to know even more about object-oriented software development.

Microsoft recognized the popularity of and satisfaction with UNIX shells and decided to merge the pipelining concept of UNIX shells with the .NET Framework. The goal was to develop a new shell that was simple to use but nearly as robust as a .NET program. The result: WPS.

In the first beta version, the new shell was presented under the code name Monad at the Professional Developer Conference (PDC) in October 2003 in Los Angeles. After the intermediate names Microsoft Shell (MSH) and Microsoft Command Shell, the shell received its final name, PowerShell, in May 2006. The final version of WPS 1.0 was released on November 11, 2006 at TechEd Europe 2006.

NOTE The main architect of WPS 1.0 was Jeffrey Snover. He is always willing to discuss his "baby" and answer questions. At large international Microsoft technical conferences, such as the Professional Developer Conference (PDC) and TechEd, you can easily find him; he is the only person at the Microsoft booths wearing a tie.

Why Use WPS?

If you need a reason to use WPS, here it comes. Just consider the following solution for one common administrative task in both the *old* WSH and the *new* WPS.

An inventory script for software is to be provided that will read the installed MSI packages using WMI. The script will get the information from several computers and summarize the results in a CSV file (*softwareinventory.csv*). The names (or IP addresses) of the computers to be queried are read from a TXT file (*computers.txt*).

The solution with WSH (Listing 1.1) requires 90 lines of code (including comments and parameterizing). In WPS, you can do the same thing in just 13 lines (Listing 1.2). If you do not want to include comments and parameterizing, you need just one line (Listing 1.3).

Listing 1.1 Software Inventory Solution 1: WSH

```
Option Explicit

' --- Settings
Const InputFileName = "computers.txt"
Const OutputFileName = "softwareinventory.csv"
```

(continues)

Listing 1.1 Software Inventory Solution 1: WSH *(continued)*

```
Const Query = "SELECT * FROM Win32_Product where not
➥Vendor like '%Microsoft%'"

Dim objFSO                ' Filesystem Object
Dim objTX                 ' Textfile object
Dim i                          ' Counter
Dim Computer         ' Current Computer Name
Dim InputFilePath    ' Path for InputFile
Dim OutputFilePath   ' Path of OutputFile

' --- Create objects
Set objFSO = CreateObject("Scripting.FileSystemObject")

' --- Get paths
InputFilePath = GetCurrentPath & "\" & InputFileName
OutputFilePath = GetCurrentPath & "\" & OutputFileName

' --- Create headlines
Print        "Computer" & ";" & _
      "Name" & ";" & _
     "Description" & ";" & _
     "Identifying Number" & ";" & _
     "Install Date" & ";" & _
     "Install Directory" & ";" & _
     "State" & ";" & _
     "SKU Number" & ";" & _
     "Vendor" & ";" & _
     "Version"

' --- Read computer list
Set objTX = objFSO.OpenTextFile(InputFilePath)

' --- Loop over all computers
Do While Not objTX.AtEndOfStream
     Computer = objTX.ReadLine
     i = i + 1
     WScript.Echo "=== Computer #" & i & ": " & Computer
       GetInventory Computer
Loop

' --- Close Input File
```

```
objTX.Close

' === Get Software inventory for one computer
Sub GetInventory(Computer)

Dim objProducts
Dim objProduct
Dim objWMIService

' --- Access WMI
Set objWMIService = GetObject("winmgmts:" &_
    "{impersonationLevel=impersonate}!\\" & Computer &_
    "\root\cimv2")
' --- Execeute WQL query
Set objProducts = objWMIService.ExecQuery(Query)
' --- Loop
For Each objProduct In objProducts
    Print _
    Computer & ";" & _
    objProduct.Name & ";" & _
    objProduct.Description & ";" & _
    objProduct.IdentifyingNumber & ";" & _
    objProduct.InstallDate & ";" & _
    objProduct.InstallLocation & ";" & _
    objProduct.InstallState & ";" & _
    objProduct.SKUNumber & ";" & _
    objProduct.Vendor & ";" & _
    objProduct.Version
Next
End Sub

' === Print
Sub Print(s)
Dim objTextFile
Set objTextFile = objFSO.OpenTextFile(OutputFilePath, 8, True)
objTextFile.WriteLine s
objTextFile.Close
End Sub

' === Get Path to this script
Function GetCurrentPath
GetCurrentPath = objFSO.GetFile (WScript.ScriptFullName).ParentFolder
End Function
```

Listing 1.2 Software Inventory Solution 2: WPS Script

```
# Settings
$InputFileName = "computers.txt"
$OutputFileName = "softwareinventory.csv"
$Query = "SELECT * FROM Win32_Product where not
➥Vendor like '%Microsoft%'"

# Read computer list
$Computers = Get-Content $InputFileName

# Loop over all computers and read WMI information
$Software = $Computers | foreach { get-wmiobject -query $Query -
computername $_ }

# Export to CSV
$Software | select Name, Description, IdentifyingNumber, InstallDate,
➥InstallLocation, InstallState, SKUNumber, Vendor, Version |
➥export-csv $OutputFileName -notypeinformation
```

Listing 1.3 Software Inventory Solution 3: WPS Pipeline Command

```
Get-Content "computers.txt" | Foreach {Get-WmiObject -computername
➥$_  -query "SELECT * FROM Win32_Product where not
➥Vendor like '%Microsoft%'" } |  Export-Csv "Softwareinventory.csv"
➥-notypeinformation
```

Downloading and Installing WPS

Windows Server 2008 is the first operating system that includes WPS on the DVD. However, it is an additional feature that can be installed through Add Feature in the Windows Server 2008 Server Manager.

WPS can be downloaded (see Figure 1.1) and installed as an add-on to the following operating systems:

- Windows XP for x86 with Service Pack 2
- Windows XP for x64 with Service Pack 2
- Windows Server 2003 for x86 with Service Pack 1

- Windows Server 2003 for x64 with Service Pack 1
- Windows Server 2003 for Itanium with Service Pack 1
- Windows Vista for x86
- Windows Vista for x64

Note that WPS is not included in Windows Vista, although Vista und WPS were released on the same day. Microsoft decided not to ship any .NET-based applications with Vista. Only the .NET Framework itself is part of Vista.

POWERSHELL DOWNLOAD PAGE www.microsoft.com/windowsserver2003/technologies/management/powershell/download.mspx

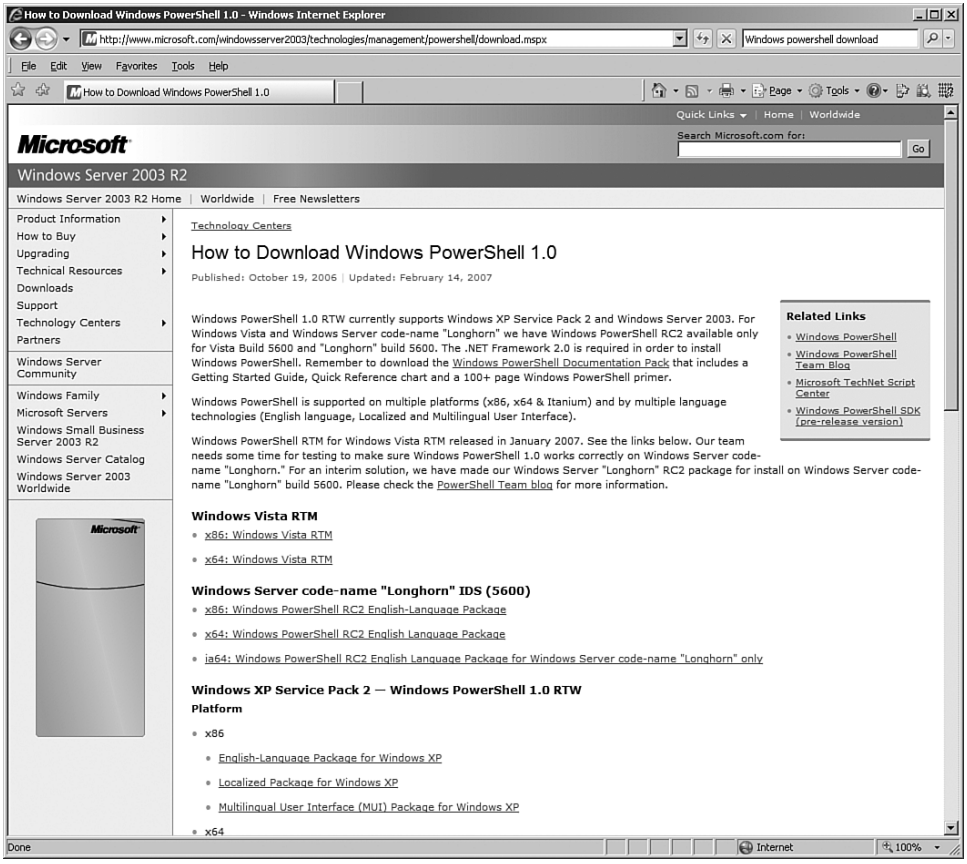

Figure 1.1 WPS download website

WPS requires that .NET Framework 2.0 or later be installed before running WPS setup. Because Vista ships with .NET Framework 3.0 (which is a true superset of 2.0), no .NET installation is required for it. However, on Windows XP and Windows Server, you must install .NET Framework 2.0, 3.0, or 3.5 first (if they are not already installed by another application).

MICROSOFT .NET FRAMEWORK 3.0 REDISTRIBUTABLE PACKAGE
www.microsoft.com/downloads/details.aspx?FamilyId=10CC340B-F857-4A14-83F5-25634C3BF043&displaylang=en

The setup routine installs WPS to the directory *%systemroot%\system32\WindowsPowerShell\V1.0* (on 32-bit systems) or *%systemroot%\Syswow64\WindowsPowerShell\V1.0* (for 64-bit systems). You cannot change this folder during setup.

TIP If for any reason you want to uninstall WPS, note that WPS is considered a software update to the Windows operating system (that is, not a normal application). Therefore, in the Add or Remove Programs control panel applet, it is not listed as a program; instead, it is listed as an update called Hotfix for Windows (KB *x*). The Knowledge Base (KB) number varies on different operating systems. However, you can identify WPS installation in the list by its icon (see Figure 1.2). On Windows XP and Windows Server 2003, you must check the Show Updates check box to see the WPS installation.

Taking WPS for a Test Run

This section includes some commands to enable you to try out a few WPS features. WPS has two modes, interactive mode and script mode, which are covered separately.

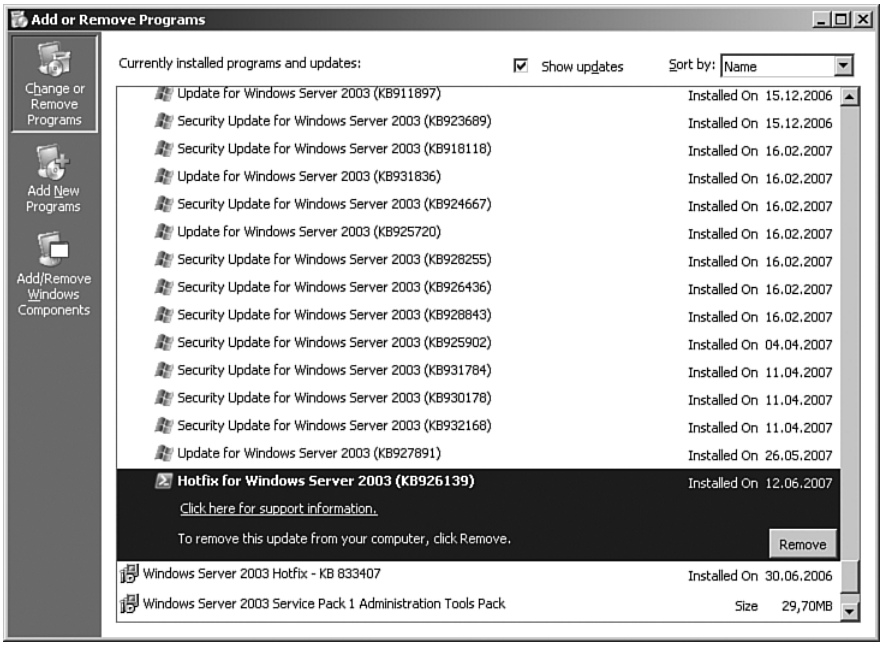

Figure 1.2 The uninstall option for WPS is difficult to find. (This screenshot is from Windows Server 2003.)

WPS in Interactive Mode

First, you'll use WPS in interactive mode.

Start WPS. An empty WPS console window will display (see Figure 1.3). At first glance, you might not see much difference between it and the traditional Windows console. However, there is much more power in WPS, as you will soon see.

At the command prompt, type **`get-process`** and then press the Return key. A list of all running processes on your local computer will display (see Figure 1.4). This was your first use of a simple WPS commandlet.

NOTE Note that the letter case does not matter. WPS does not distinguish between uppercase and lowercase letters in commandlet names.

```
Windows PowerShell
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. All rights reserved.

PS C:\Documents\hs> _
```

Figure 1.3 Empty WPS console window

```
Windows PowerShell

PS C:\Documents\hs> get-process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     90       5     6504       7752    79     0.03   1336 Bildschirmpausenreminderdienst
    107       3     1236       1260    30     0.06   4232 cidaemon
     82       3     1260       1088    30     0.06   4712 cidaemon
     82       3     1336       1088    30     0.06   4892 cidaemon
    414       0     2764        636    43     0.95   1376 cisvc
   1002       9     2460       6360    29     5.41   1652 csrss
     69       3      496       2876    16     0.09   3936 ctfmon
     29       1      368       1612    16     0.00   1408 DefWatch
    214       4     2364       4088    44     0.16   1468 dlpsp
    130      10     1436       3000    30     0.02   2244 dlpudnt
     57       2      660       2172    10     0.00   1416 dlsdbnt
     96     126     6532       6148   316     0.03   2420 exmgmt
    563      17    21588      19532   129    28.61   2276 explorer
    226      14    13120      20856    81    35.30   4004 FolderShare
    100       5     2032        672    65     0.00   2640 GoogleToolbarNotifier
    106       4     1732       5040    44     0.09   3004 GrooveMonitor
      0       0        0         28     0        -      0 Idle
    696      19    32596       6396   178     2.91   4224 iexplore
    643      63    17816      17040   118     0.69   1504 instinfo
     63       3      520       2000    17     0.23   1524 ISRService
     33       2      964       3300    27     0.02   4072 Launcher
    712      26     9504      11588    57     1.52   1068 lsass
     55       3     3588       7404    34     0.27   3852 Matrox.PowerDesk SE
    304       9    24688      24204   156     1.59   3996 Matrox.PowerDesk.PDeskNet
     29       1      204       1500    14     0.02   1572 Matrox.PowerDesk.Services
     29       1      204       1500    14     0.00   1580 Matrox.PowerDesk.Services
     92       3     1048       3508    30     0.03   1604 mdm
    418       8     9484        600    74     0.19    816 MOMHost
    235       5     2996        460    44     0.41    856 MOMHost
    754       9     6140       8376    60     1.06   1604 MOMService
    251     131     4636       8000    47     0.13   2624 msgsvc
     51       2     1900       4040    24     0.39   5536 msconfsvu
    165      16     1752       4384    25     0.05   1148 mstdc
    281      15     9472        820    75     0.28   1076 ntfrs
     86       3     2220       4096    33     0.17   3680 NvMixerTray
     83       3     1132       4304    33     0.05   3640 nvraidservice
    232       6    29212      28372   133     3.33   5884 powershell
    136       5     2504       5000    44     0.09   3964 rapimgr
    329      10    25620      28688   110     0.63    444 Rtvscan
    465      13     2300       4592    73    22.00   1056 services
     10       1      164        504     4     0.06   1352 smss
    385      14    11544      11492   155     4.17   5284 Snagit32
    287       9     6100       8612    54     0.36   1112 spoolsv
     61       3      716       2400    14     0.00   1632 sqlbrowser
    335       9    37472       1392  1493     0.30   1708 sqlservr
     70       2     1412       4040    20     0.00   1760 sqlwriter
    372      21     1748       4316    24     1.50    344 svchost
   1169      58    21616      29404   191    49.17    400 svchost
    156       7     3980       4684    41     0.03    456 svchost
    175       6     1104       3360    22     0.02    536 svchost
     39       1      300       1204     7     0.00   1332 svchost
     56       2      544       2120    16     0.00   1444 svchost
     85       3     1072       3220    21     0.09   2028 svchost
    158      11     2968       5980    39     0.11   2084 svchost
    122       4     3596       5180    26     0.05   2200 svchost
    155       7     4156       7020    35     0.06   2652 svchost
     77       3     2144       4052    19     0.02   2672 svchost
    163       5     2420       4448    56     0.08   3032 svchost
    221       8     2896       4004    32     0.09   3800 svchost
   2436       0        0        236     2    20.47      4 System
     35       2      656       2916    26     0.02   4620 TscHelp
     91       5    26972      29004    68    18.66   2460 TSVNCache
    106       4     3096       7344    30     0.09   2112 TTTvRc
     64       2     1876       3420    24     0.00   3896 unsecapp
    103       3     2324       4180    36     0.14   2476 VPTray
    251      10    24140      33600   146     5.61   5320 vJup
    113       5     1352       4868    43     0.19   1868 vcscomm
    594      86     7568       3144    53     1.30   1740 winlogon
    589      22    50268      82920   400    18.36   4484 WINWORD
    115       3    14004       4344    25     0.05   2136 wmiprvse
    196       5     4212       6540    42     2.06   2464 wmiprvse
    200       5     2392       6996    43     1.95   2772 wmiprvse

PS C:\Documents\hs>
```

Figure 1.4 The `Get-Process` commandlet output

At the command prompt, type **`get-service i*`**. A list of all installed services with a name that begins with the letter *I* on your computer will

display (see Figure 1.5). This was your first use of a commandlet with parameters.

Figure 1.5 A filtered list of Windows services

Type **`get-`** and then press the Tab key several times. You will see WPS cycling through all commandlets that start with the verb *get.* Microsoft calls this feature *tab completion.* Stop at `Get-Eventlog`. When you press Enter, WPS prompts for a parameter called LogName (see Figure 1.6). LogName is a required parameter. After typing **`Application`** and pressing Return, you will see a long list of the current entries in your Application event log.

Figure 1.6 WPS prompts for a required parameter.

The last example in this section introduces you to the pipeline features of WPS. Again, we want to list entries from a Windows event log, but this time we want to get only some entries. The task is to get the most recent ten events that apply to printing. Enter the following command, which consists of three commandlets connected via pipes (see Figure 1.7):

```
Get-EventLog system | Where-Object { $_.source -eq "print" }
➥ | Select-Object -first 10
```

Note that WPS seems to get stuck for a few seconds after printing the first ten entries. This is the correct behavior because the first commandlet

(`Get-EventLog`) will receive all entries. The filtering is done by the subsequent commandlets (`Where-Object` and `Select-Object`). Unfortunately, `Get-EventLog` has no included filter mechanism.

```
Windows PowerShell
PS C:\Documents\hs>
PS C:\Documents\hs> Get-EventLog system | where-object { $_.source -eq "print" } | select-object -first 10

Index Time          Type Source                EventID Message
----- ----          ---- ------                ------- -------
...27 Jun 12 19:02  Info Print                     10 Document 2, Microsoft Word - TY25_1.doc owned by hs was print...
...48 Jun 12 14:24  Info Print                     10 Document 26, infas_Angebot_A-27715088.pdf owned by hs was pri...
...46 Jun 12 14:19  Info Print                     10 Document 25, http://reiseauskunft.bahn.de/bin/query.exe/dn?ld...
...45 Jun 12 14:17  Info Print                     10 Document 24, http://reiseauskunft.bahn.de/bin/query.exe/dn?ld...
...42 Jun 12 13:57  Info Print                     10 Document 23, http://reiseauskunft.bahn.de/bin/query.exe/dn?ld...
...41 Jun 12 13:48  Info Print                     10 Document 22, Microsoft Office Outlook - Memoformat owned by h...
...39 Jun 12 13:47  Info Print                     10 Document 21, Microsoft Office Outlook - Memoformat owned by h...
...33 Jun 12 13:10  Info Print                     10 Document 20, Microsoft Word - 2422 Fachlektorat.doc owned by ...
...32 Jun 12 13:10  Info Print                     10 Document 19, Microsoft Office Outlook - Memoformat owned by h...
...31 Jun 12 13:09  Info Print                     10 Document 18, Microsoft Office Outlook - Memoformat owned by h...
```

Figure 1.7 Filtering event log entries

WPS in Script Mode

Now it's time to try out PowerShell in script mode and incorporate a WPS script. A WPS script is a text file that includes commandlets/elements of PowerShell Script Language (PSL). The script in this example creates a new user account on your local computer.

Open Windows Notepad (or any other text editor) and enter the following lines of script code (which consists of comments, variable declarations, COM library calls, and shell output):

Listing 1.4 Create a User Account

```
### PowerShell Script
### Create local User Acount

# Variables
$Name = "Dr. Holger Schwichtenberg"
$Accountname = "HolgerSchwichtenberg"
$Description = "Author of this book / Website: www.windows-scripting.com"
$Password = "secret+123"
$Computer = "localhost"

"Creating User on Computer $Computer"
```

```
# Access to Container using the COM library
➥"Active Directory Service Interface (ADSI)"
$Container = [ADSI] "WinNT://$Computer"

# Create User
$objUser = $Container.Create("user", $Accountname)
$objUser.Put("Fullname", $Name)
$objUser.Put("Description", $Description)
# Set Password
$objUser.SetPassword($Password)
# Save Changes
$objUser.SetInfo()

"User created: $Name"
```

Save the text file with the name **`createuser.ps1`** into the directory *c:\temp*. Note that the file extension must be `.ps1`.

Now start WPS. Try to start the script by typing **`c:\temp\createuser.ps1`**. (You can use tab completion for the directory and filenames.) This attempt will fail because script execution is, by default, not allowed in WPS (see Figure 1.8). This is not a bug; it is a security feature. (Remember the Love Letter worm for WSH?)

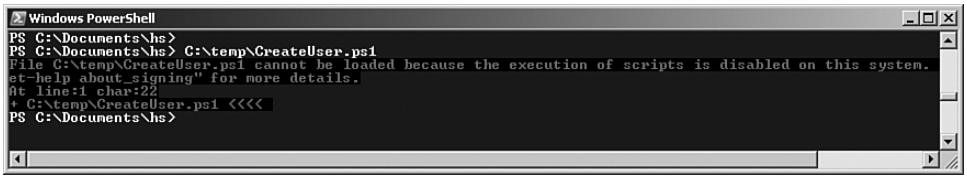

Figure 1.8 Script execution is prohibited by default.

For our first test, we will weaken the security a little bit (just a little). We will allow scripts that reside on your local system to run. However, scripts that come from network resources (including the Internet) will need a digital signature from a trusted script author. Later in this book you learn how to digitally sign WPS scripts. You also learn to restrict your system to scripts that you or your colleagues have signed.

To allow the script to run, enter the following:

```
Set-ExecutionPolicy remotesigned
```

Then, start the script again (see Figure 1.9). Now you should see a message that the user account has been created (see Figure 1.10).

Figure 1.9 Running your first script to create a user account

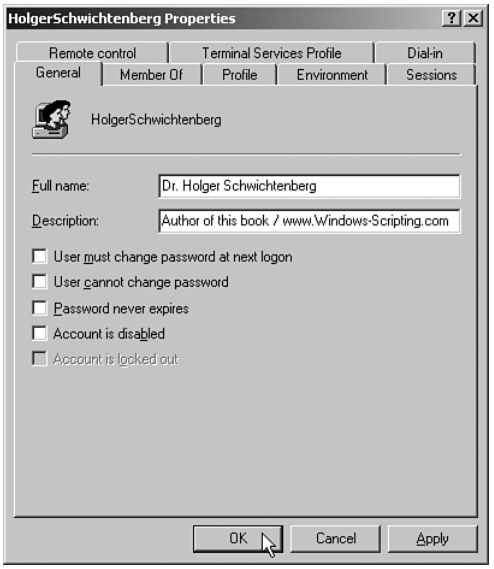

Figure 1.10 The newly created user account

Downloading and Installing PowerShell Community Extensions

WPS 1.0 includes only 129 commandlets. You might ask why I wrote *only.* You will notice soon that the most important commandlets are those with the verbs `get` and `set`. And the number of those commandlets is quite small compared to the large number of objects that Windows operating systems provide. All the other commandlets are, more or less, related to WPS infrastructure (for example, filtering, formatting, and exporting).

PowerShell Community Extensions (PSCX) is an open source project (see Figure 1.11) that provides additional functionality with commandlets such as `Get-DhcpServer`, `Get-DomainController`, `Get-MountPoint`, `Get-TerminalSession`, `Ping-Host`, `Write-GZip`, and many more. Microsoft leads this project, but any .NET software developer is invited to contribute. New versions are published on a regular basis. At the time of this writing, version 1.1.1 is the current stable release.

DOWNLOAD POWERSHELL COMMUNITY EXTENSIONS
www.codeplex.com/PowerShellCX

PSCX is provided as a setup routine that should be installed after WPS has been installed successfully.

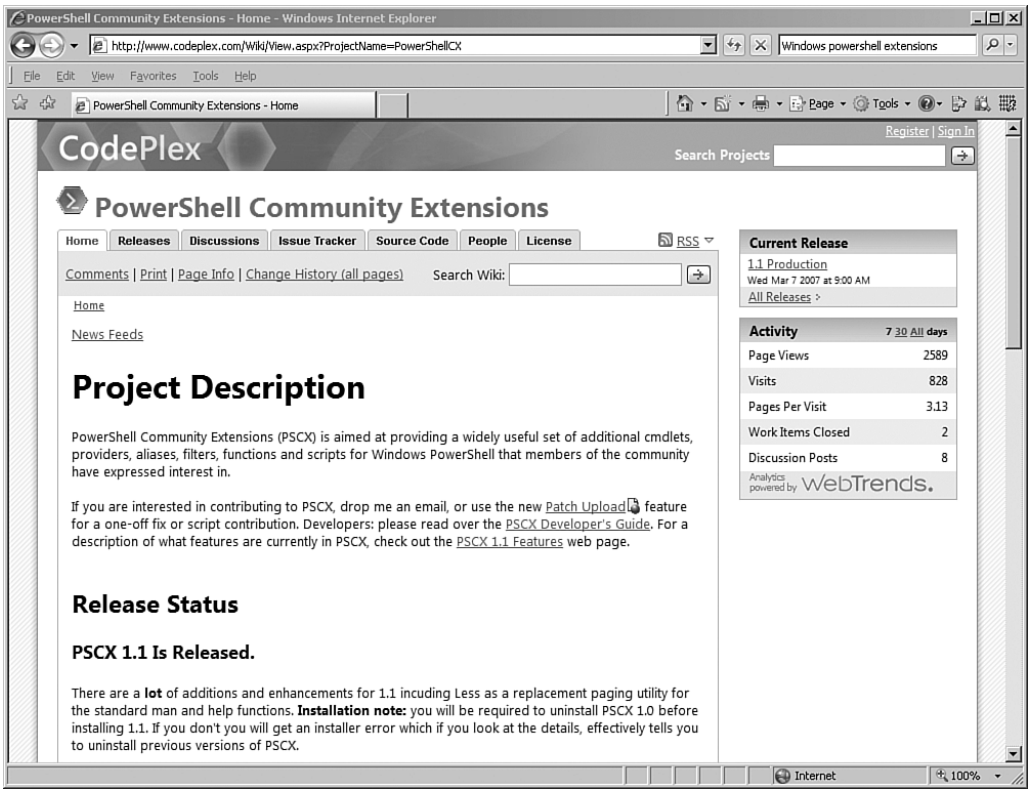

Figure 1.11 PowerShell Community Extension website

You can incorporate additional functionality of PSCX into WPS by using a profile script (see Figure 1.12). Just copy this profile script to your *My Documents/Windows PowerShell* directory, if you want, during PSCX setup. As a beginner, you should use this option.

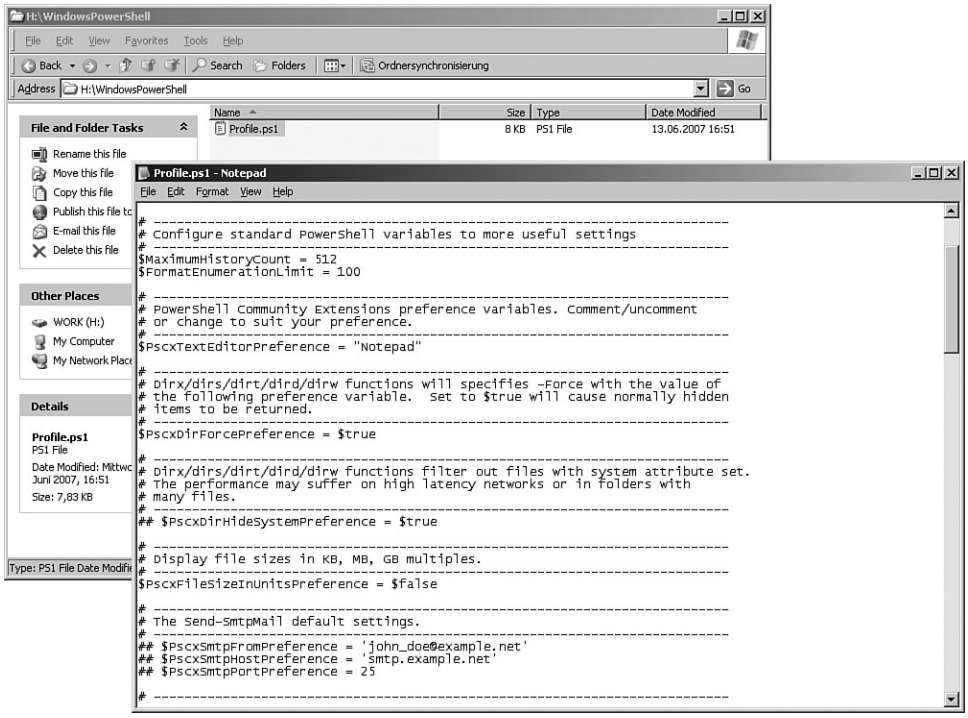

Figure 1.12 The PSCX profile script that was created during PSCX setup

Testing the PowerShell Extensions

The installation of PSCX changes the WPS console just a bit. Instead of the current path, the prompt now contains a counter. However, the path does display in the window's title.

Start WPS and type **`Get-DomainController`** (if your computer is a member of an Active Directory) or test PSCX by using **`Ping-Host`** with any computer on your network (see Figure 1.13).

```
PoSh C:\WINDOWS
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. All rights reserved.

1# Get-DomainController


ServerName     : E02
DnsHostName    : E02.IT-Visions.local
Site           : Default-First-Site
Domain         : IT-Visions.local
DN             : CN=E02,OU=Domain Controllers,DC=IT-Visions,DC=local
GlobalCatalog  : True



2# ping-host www.IT-Visions.de
Pinging www.it-visions.de with 32 bytes of data:
    Reply from 195.234.228.60 bytes=32 time=16ms TTL=117
    Reply from 195.234.228.60 bytes=32 time=15ms TTL=117
    Reply from 195.234.228.60 bytes=32 time=16ms TTL=117
    Reply from 195.234.228.60 bytes=32 time=15ms TTL=117

Ping statistics for www.it-visions.de:
    Packets: Sent = 4 Received = 4 (0% loss)
    Approximate round trip time: min = 15ms, max = 16ms, avg = 15ms



3#
```

Figure 1.13 Testing `Get-DomainController` and `Ping-Host`

Downloading and Installing the PowerShellPlus

Unfortunately, Microsoft does not provide a script editor for WPS yet. However, a few third-party editors support WPS (see Chapter 9, "PowerShell Tools"). Throughout this book, we use PowerShellPlus Editor, which is free for noncommercial use.

A previous editor called PowerShell IDE from the same author was free even for commercial use. However, PowerShell IDE never made it to a final release and was discontinued.

The PowerShellPlus Editor is part of PowerShellPlus. PowerShellPlus consists of the editor and a console that provides IntelliSense while using the PowerShell interactively.

POWERSHELLPLUS WEBSITE www.powershell.com

PowerShellPlus does not need any setup. It is a true .NET application with XCopy deployment. You just unpack the ZIP file to the directory of your choice and start the `PowerShellPlus.exe` that is part of the package.

Testing the PowerShell Editor

The PowerShellPlus has, according to the WPS console, two modes: an interactive mode and a script mode (see Figure 1.14). After starting the PowerShellPlus, you will see the interactive mode. You can use any commandlet (or pipeline). When you press Return, the commandlet is executed, and the result displays in the same window. The handy feature is the IntelliSense. If you enter **`Get-P`**, you will see a drop-down list of the available commandlets that start with these letters.

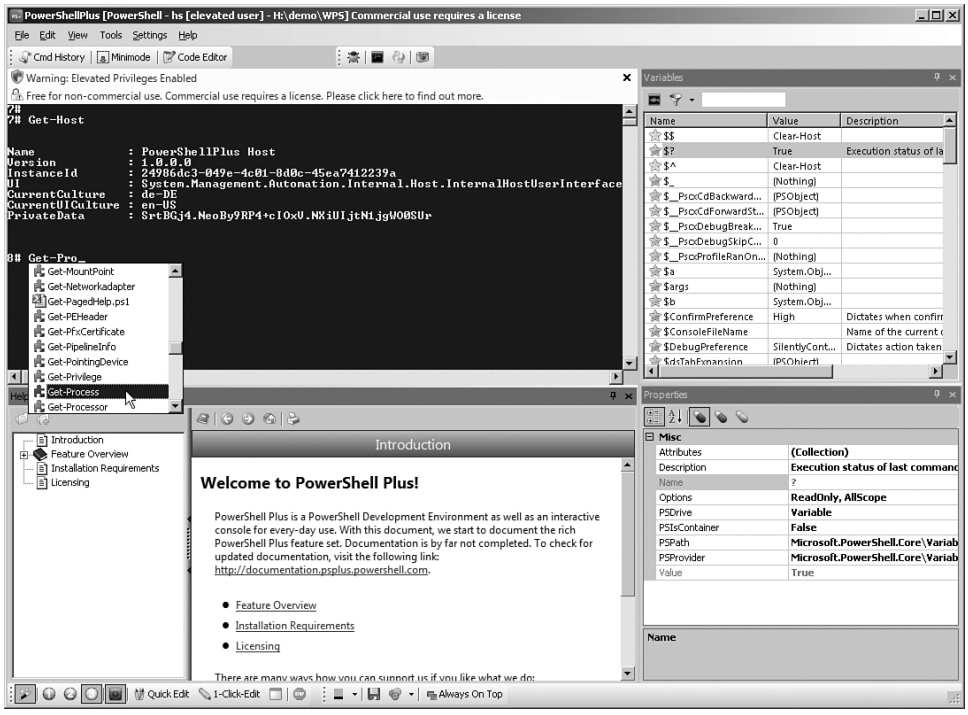

Figure 1.14 WPS IDE in interactive mode

To use the PowerShellPlus in script mode, click Code Editor and create a new script file (New/PowerShell Script) or open an existing script PS1 file (Open). Now open the script file `CreateUser.ps1` that you created earlier. You will see line numbers, and you will encounter the same IntelliSense features that you have in interactive mode. To run the script,

click the Run symbol in the toolbar (see Figure 1.15). The result will display in the interactive Windows in the background.

WARNING Make sure the user account does not exist before running the script. Otherwise the script will fail with the error "The account already exists."

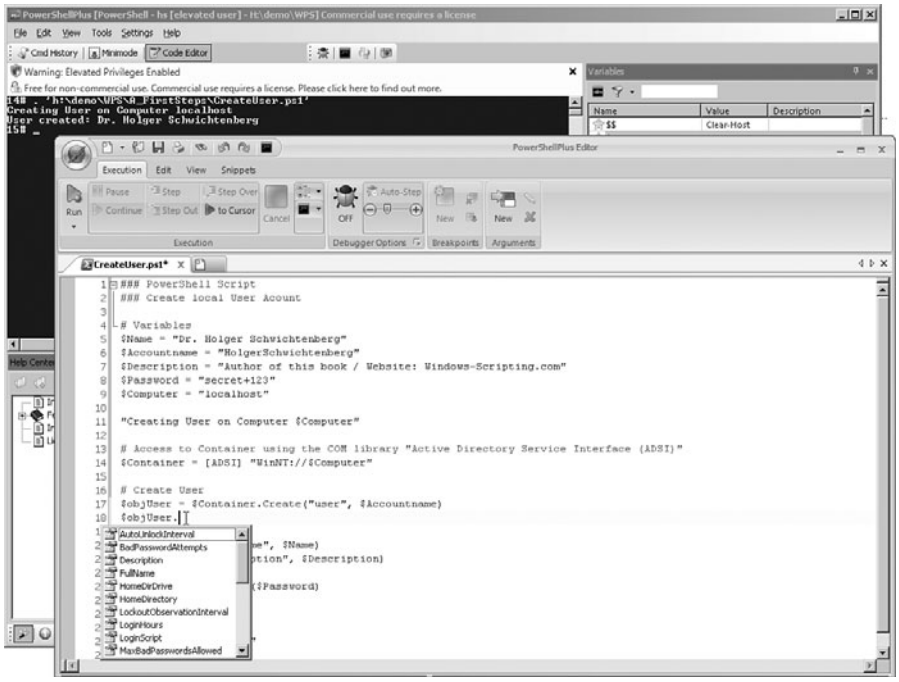

Figure 1.15 WPS IDE in script mode

Another great feature is debugging. Place the cursor on any line in your script and click the Debugging icon. Next, go to any line and press F9. This creates a red circle next to that line, called a *breakpoint.* Now run the script. You will see the PowerShellPlus Editor executing the script in slow motion, marking the current line yellow and stopping at the line with the breakpoint (see Figure 1.16). In the Variables Inspector window, you can inspect the current value of all variables. In the interactive window, you can type any WPS command that will be executed within the current context. That is, you can interactively access all script variables. To continue the script, press F8 or click the Continue icon in the toolbar.

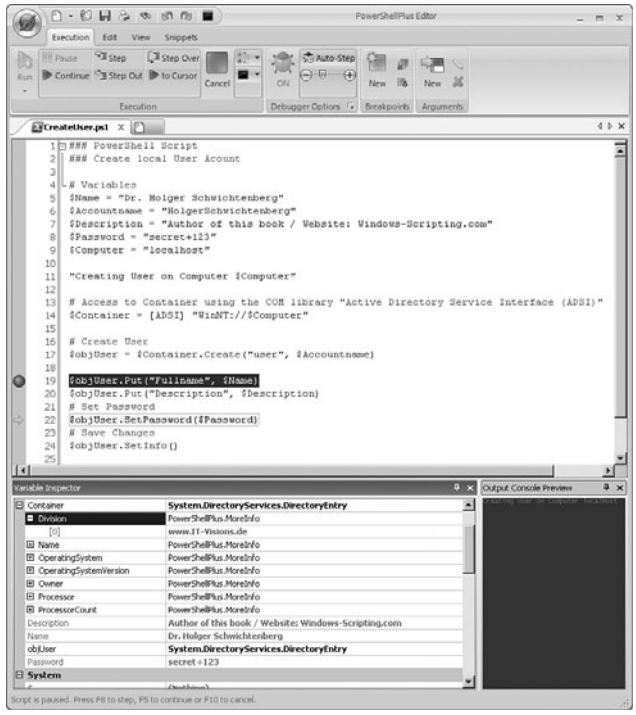

Figure 1.16 Script debugging with the WPS IDE

Code snippets are also a nice feature of the PowerShellPlus. In a script file, click Snippet/Insert on the toolbar or select Insert Snippet in the context menu in the main Editor window. You will be able to select a snippet. You can create you own snippets with the PowerShellPlus (via Snippets/New on the toolbar).

Summary

Windows PowerShell is a new .NET-based environment for scripting and is an interactive command-line shell. WPS is an optional feature on Windows Server 2008 and an add-on for Windows XP, Vista, and Server 2008. Commands in WPS are called commandlets. The PSCX extends WPS with additional commandlets.

The PowerShellPlus is an alternative shell for WPS commands and an editor for WPS scripts.

In the next chapter, you learn much more about commandlets and pipelines. You also learn how to get help if you are seeking a command or the available options for a commandlet.

INDEX

B–C

D

H

J–K

L

M

X–Z