MANNING PUBLICATIONS

Conquering administrative challenges with PowerShell and WMI

Green Paper from

PowerShell and WMI

Richard Siddaway
Softbound print: 2012 | 450 pages
ISBN: 9781617290114

This green paper is taken from the book ***PowerShell and WMI*** *from Manning Publications. The author explains how PowerShell and WMI provide a set of tested techniques that will enable you to administer your Windows environment in a faster and easier way.*

Ask any Windows administrator about their biggest problems and, somewhere in the list, usually near the top, will be some reference to too much work and not enough time to do it. They know that automation is possible and are at least aware of some of the technologies such as Windows Management Instrumentation (WMI) and PowerShell that could solve their problems, but they don't have the time to spend investigating the technologies. It is also possible that they have looked at WMI or PowerShell and decided it was too hard for them. They miss out on the possibilities that automation provides to reduce their workload and achieve more.

PowerShell and WMI provide a set of tested techniques that will enable you to administer your Windows environment in a faster and easier way. PowerShell is Microsoft's automation engine that provides easy-to-use access to the rich management toolset available in WMI. You will be able to automate many of the standard tasks that currently consume too much of your attention. This will free up time to do the more interesting things that just can't be fitted into the normal working day.

The first thing we need to do is define the problem we are trying to solve. There are a number of issues affecting any Windows environment of significant size:

- Number of systems
- Rising infrastructure complexity
- Rate of change

The second part of the chapter shows why PowerShell and WMI provide a good toolset for solving these problems. This involves investing a little time to learn PowerShell and how to get the most out of it, especially when using WMI. Automation is the key to making our life as administrators easier. PowerShell and WMI make automation an economic solution in terms of the benefits we can achieve compared with the investment in learning to use the technologies.

Let's jump into the discussion with a look at the responsibilities of a modern Windows administrator and the problems administrators are facing.

Administrative challenges

Administrators are very busy people. They seem to be in a continual spiral of always being asked to do more with fewer resources. The graph in figure 1 illustrates the ever-decreasing cost of hardware, in real terms. A recent purchase of mine illustrates this point. I acquired a laptop with a quad core processor (hyper-threading allows Windows to see 8 cores) and 16 GB of RAM as a mobile lab. A few years ago, a machine of that specification was a server not a laptop!

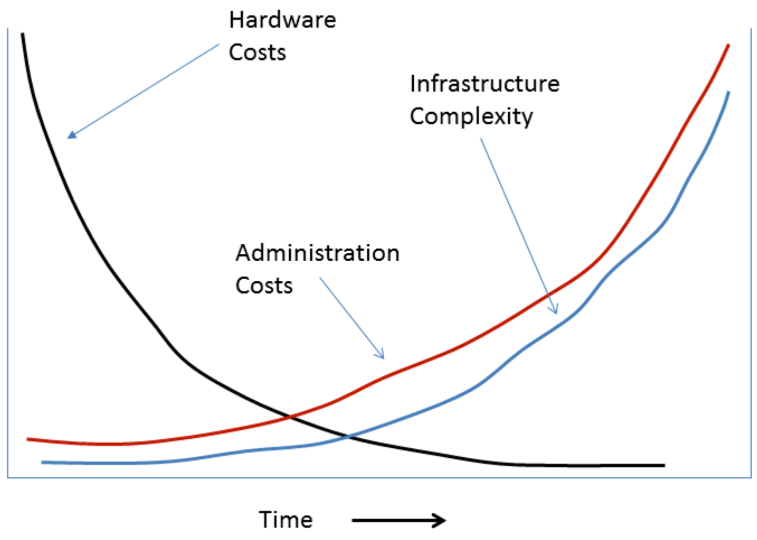


Figure 1 The relationship between the dwindling hardware costs, the infrastructure complexity, and the cost of administering the evolving infrastructure.

The same is true in the server market—quad core processors and a lot of relatively cheap memory mean that we can afford to run applications and business processes that were previously only considered by large corporations with huge budgets.

This leads directly to the other components of the graph showing the steep rise in complexity and the even faster rise in administration costs. The continual upward growth of infrastructure complexity and cost is not sustainable. PowerShell and WMI will help you break out of this growth curve.

Now, we need to examine the problem in a little more depth—where the complexity and the cost of administration come from.

Too many machines

It may seem to be an odd way to look at infrastructure, but do you really need every server you have created? Many, if not most, organizations have too many servers. This has come about for a number of reasons:

- The decreasing cost of hardware generates a belief that it's easier to add a new server than to use an existing one.
- Departmental or project-based purchasing causes issues of server ownership and an unwillingness of departments or projects to share resources.
- The one application–one server rule has been regarded as good practice because it separates applications so that a problem in one doesn't affect other applications. This may still be valid for business-critical applications but not necessarily for second- or third-line applications. It is definitely not required for testing and training versions.
- Lack of controls in IT prompts departments and projects to introduce systems that IT doesn't know about until they hit production.

An administrator's workload increases faster than the rate of machine increase due to the time spent switching between machines and the additional complexity each machine and its supported applications bring to the environment.

Virtualization is one of the hot IT topics, with many organizations virtualizing at least part of their server estate. The advantages of virtualization include:

- Reduced numbers of physical servers.
- Reduced requirement for data center facilities including space, power, and air conditioning.
- Increased use of physical assets, giving better return on investment.

The organization as a whole benefits from virtualization but the administrator's load is increased. Say, an administrator had 100 servers to administer before virtualization. If we use four physical hosts and virtualize our 100 servers, the administrator now has 104 servers to administer. The complexity may increase as well because the virtualization platform may introduce a different operating system into the environment. The increase in the number of machines also means that there will be more changes as the environment evolves.

Too many changes

Change can be viewed as an administrator's worst headache. Unfortunately, our environments aren't static:

- Operating system and application patches are released on a regular basis.
- New versions of software are released.
- Storage space is re-adjusted to match usage patterns.
- Application usage patterns force hardware upgrades.
- Virtualization and other disruptive new technologies change the way we create and configure our environments.

This level of activity multiplied across the 10s, 100s, or even 1000s of machines in our environment is building on top of the day-to-day activities such as monitoring and backup.

The situation is not supportable in the long run. Organizations can't absorb the ever-increasing administration costs, and today's economic realities prevent other mechanisms, such as increased revenue, from providing an escape. The situation has to be resolved by reducing the cost of administration. This is hampered by the fact that many changes bring new technologies into the environment without ensuring they are supportable.

Complexity and understanding

Complexity is the real problem in many cases. It can arise due to a number of causes:

- Multiple operating systems.
- Many different types of applications, for example, databases, email, Active Directory, or web-based applications.
- Having many machines do the same or similar roles.

This problem is often compounded by an incomplete knowledge and skill set on the administrator's part. Too many times a project comes along, introduces a new technology, and expects us to immediately pick it up and manage the systems. Do we have the skills? Do we have the time to learn the intricacies of this new technology? Very sadly, the answer to both questions is often no.

As a result, administrators make best guess decisions on how to do things. Sometimes, if the new technology is a version change from something we already use, we will continue to employ the old methods even if there is a better way to perform the task.

This lack of skills and knowledge leads to mistakes. These mistakes cost money, often in terms of lost revenue for our organizations. This puts more pressure on the administrators and leads to a diminished trust from the business. IT is often then excluded from discussions about new technologies until it is too late and the cycle takes another spiral downwards.

Not only do projects introduce major changes, we also have a host of minor changes to keep our environments secure and running smoothly.

The way forward

The way to overcome these issues is to introduce automation. Get the machine to do the mundane, repetitive work—that's what we invented them for!

Automation means many things to many people. Notice the hierarchy of automation activity in figure 2.

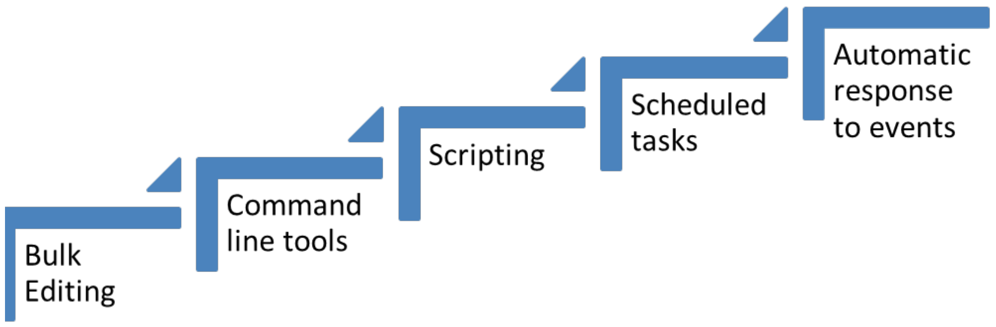


Figure 2 Hierarchy of automation activity

The question that needs to be answered by every organization is, "Where do I get the most benefit?" My usual response is that it depends on what you are trying to achieve and where you are now. I know of a number of organizations that are quite happy using the standard Windows tools and a few bulk editing tools. Others are attempting to schedule everything or even create automated responses to events.

My personal take is that automation, for most organizations, is a mixture of command line tools, scripting, and scheduled tasks. That leads onto the second big question, "How do I automate my administrative tasks." PowerShell provides a set of command line tools (called cmdlets) that can be used interactively. As commands we enter become longer and more ambitious, there is a natural progression into scripting. One of the great strengths of PowerShell is that we can use exactly the same commands in a script or at the command prompt so everything we have learned is still usable.

PowerShell by itself is a wonderful tool (OK—yes I am fanatical about it) but we can take it a stage further and layer the use of WMI on top. This opens a standards-based management toolset that we can use on local and remote machines. The scripts can be run interactively or scheduled to run at a specific time. Guess how we schedule the scripts? By using PowerShell and WMI, of course. Before we get into those delights, let's have look at automation in general.

Automation

Some would argue that, since we are using PowerShell, we could do much of our work from the command line. In fact, listing 1 is a single line of PowerShell. The benefit of scripting is that we can reuse the code and save even more time by not having to rewrite the code each time we want to use it. This topic is covered in depth in chapter 4 of PowerShell in Practice (Manning 2010).

This is best explained through an example. You need to determine the free space on the C: drive of a number of machines in your environment. One way is to go to the data center—we will be kind and assume they are all in the same data center—and log onto the console of each machine. You will then need to open Windows Explorer or another tool and find the free space on the C: drive. Write down the answer and repeat for the next machine on the list.

A slightly easier option is to use Windows' Remote Desktop functionality to connect to each machine. You have to follow the manual process of obtaining the information. This has the advantage of not having to move from your desk but it still takes time.

My favorite solution is to use a small piece of PowerShell, as shown in listing 1.

We start with a list of server names— these are taken from my lab setup. This list is piped into a `ForEach-Object` cmdlet (aliased as `foreach`) that calls `Get-WmiObject` for each server in the list to find the information on the logical disk used as the C: drive. We format the information and output it as a table.

Listing 1 Find free disk space

```
"dc02", "W08R2CS01", "W08f2CS02", "W08R2SQL08",
"W08R2SQL08A", "WSS08" | foreach {
    Get-WmiObject -Class Win32_LogicalDisk `
-ComputerName $_ -Filter "DeviceId='C:'" } |
Format-Table SystemName, @{Name="Free"; Expression={[math]::round($($_.FreeSpace/1GB), 2)}} -
auto
```

The free space is recalculated from bytes to GB to make the results more understandable. Notice that PowerShell understands 1 GB, as well as KB, MB, TB, and PB. Our results look like this:

```
SystemName    Free
----------    ----
W08R2CS01   119.04
W08R2CS02   118.65
W08R2SQL08   114.8
W08R2SQL08A 115.17
WSS08       111.41
DC02        118.53
```

The output can be related to the script. There are a number of enhancements that we can apply to this script:

- Put the computer names in to a csv file.
- Add the results to an Excel spreadsheet or database, so that you can see trends.
- Schedule the task to run on a periodic basis.

I use a similar script, with the first two enhancements, to report disk space trends on a regular basis for the organization I am currently working with. I now have a tool that takes seconds to run against each machine and provides information vital to my job performance. It is also quickly and easily extensible to cover other machines that may become of interest. The script took me a few minutes to write and test. There is an immediate payback every time I use it.

PowerShell is designed to provide this type of return. Jeffrey Snover, the architect of PowerShell, says that "economics determine what people do and don't do so PowerShell is designed from the ground up to make composable, high-level task oriented abstractions be the cheapest things to produce and support." The full article is available from the PowerShell Team blog at http://blogs.msdn.com/b/powershell/. A search for "semantic gap" will take you to the post.

Now we'll have a closer look at PowerShell and why it is the ideal platform for automating your administration.

PowerShell Overview

I want to show you why PowerShell is the ideal platform for automating your Windows administration. PowerShell is now on its second version. It is part of the default installation of Windows 7 and Windows Server 2008 R2. (For the server core, it is an optional install.) PowerShell 2.0 also can be installed on Windows Server 2008, Windows Server 2003, Windows Vista, and XP.

WINDOWS 2000 SUPPORT

Windows 2000 is out of support at the time of this writing. PowerShell didn't have an option to install on Windows 2000.

This level of support supplies a tool to manage all of our Windows systems. An increasing number of applications also have built-in PowerShell support. (It is a requirement for all new versions of major Microsoft products.) Adoption by third-party vendors is steadily increasing the scope of PowerShell.

PowerShell References

Bruce Payette's PowerShell in Action, Second Edition (Manning 2010) provides the most detailed coverage of PowerShell from a language perspective. *PowerShell in Practice* (Manning 2010) supplies many examples of using PowerShell to administer Windows systems.

The ability to access remote machines simplifies administration because we can automate across the Windows environment from a single administration console. This is the way that we break the curve of rising infrastructure complexity. We'll see how to achieve this after we've examined PowerShell's scope.

POWERSHELL SCOPE

The scope of administration that PowerShell supplies is very broad—from applications having built-in direct PowerShell support to community inspired and provided additions that are available for download.

A number of major applications have direct PowerShell support:

- Exchange 2007/2010 (probably the poster child of PowerShell support)
- SQL Server 2008
- SharePoint 2010
- Various members of the System Center family.

Other aspects of the Windows environment have PowerShell support available through Microsoft and/or third-party additions including:

- Active Directory
- IIS
- Clustering
- Terminal Services
- Graphical presentation tools

The availability of functionality is a good thing but, for us to get the most from it, we need to shorten the time it takes us to get it into production use. One way to achieve this is via the PowerShell community supplying sample code to shorten the development cycle. PowerShell has a very strong and productive community. This starts with the PowerShell team and extends through:

- Blogs.
- Individual code examples.
- Code repositories for community contributions.
- PowerShell additions as scripts, modules or cmdlets.
- Forums.
- Books—print and electronic.

This provides a breadth, and depth, of support and additional functionality that almost guarantees you will be able to find help for solving your problem.

.NET

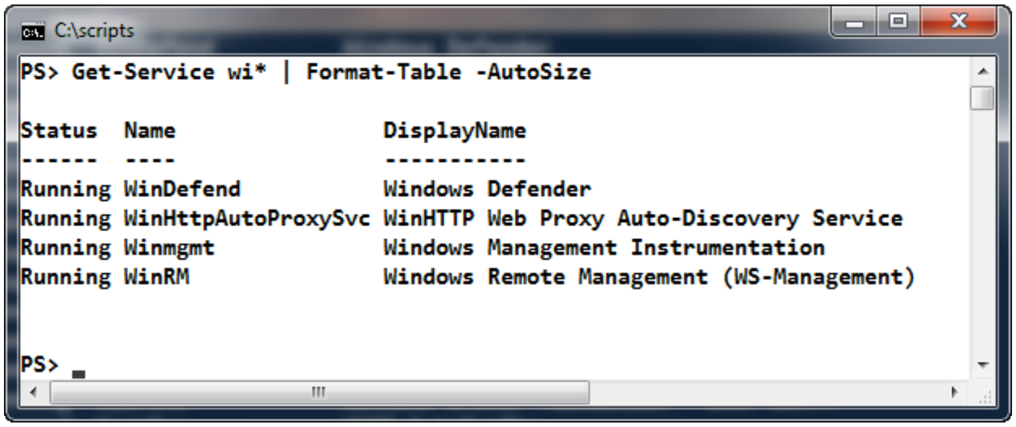
Whenever PowerShell is discussed, the fact that it is .NET based and can access most of the .NET framework is brought up. At this point, I find that the eyes of many administrators begin to glaze over and they slide down into their seats. WAKE UP!

Yes, PowerShell is .NET based and there are some really clever things that can be done by working directly with .NET code in PowerShell. The great thing about this is that you don't need to do this until *you* are ready to work at this level. There is a huge number of administration tasks we can perform without dipping our toes any further into the .NET waters. Don't forget that .NET is there when you need it, and the PowerShell community offers a lot of great examples.

WMI and .NET

It is possible to use WMI functionality through .NET code created and run in PowerShell. There are usually alternatives to using .NET in this way, and I would always choose those over a .NET-based example. I am an administrator, not a developer, and I will present solutions for administrators.

As an example, we'll consider looking at the services running on a system. A subset of the installed services is shown in figure 3.



```
PS> Get-Service wi* | Format-Table -AutoSize

Status  Name                 DisplayName
------  ----                 -----------
Running WinDefend            Windows Defender
Running WinHttpAutoProxySvc WinHTTP Web Proxy Auto-Discovery Service
Running Winmgmt              Windows Management Instrumentation
Running WinRM                Windows Remote Management (WS-Management)


PS> _
```

Figure 3 Using Get-Service to display a subset of the running services.

The `Get-Service` cmdlet (a PowerShell command) returns a list of the running services. I have restricted the output by using `wi*` to only return services starting with wi. The results are piped into a `Format-Table` cmdlet that will output the results as a nicely formatted table.

PIPELINE

The PowerShell pipeline passes .NET objects rather than text as it happens with other shells. This supplies a large measure of the technology's power.

I deliberately chose `wi*` because it demonstrates two of the services, WMI and Windows Remote Management (WinRM). (It also keeps the figure to a reasonable size.)

Underneath the hood, `Get-Service` is using a .NET class called `System.ServiceProcess.ServiceController`, which is fascinating but doesn't mean much to me without looking up the .NET documentation. The beauty is that we don't need to know this for 99.99% or more of the time. PowerShell abstracts all of this for us, and we can perform our discovery with a command that has a name that describes its purpose and is easy to use.

Breaking the curve

In figure 1, we saw that there is a continuous rise in the complexity of our organizations and in the cost of performing the administration in those organizations. We have seen that the continuous increase is not supportable in the long term and we need a way to break out of the curve.

PowerShell is one way we can break that curve. This is achieved in a number of ways:

- A set of tools to interactively administer our servers and applications
- An automation engine we can use across our whole Windows estate
- The ability to apply those concepts to a number of applications
- A number of remote administration engines that enable multiple machines to be administered with a single command
- Asynchronous and scheduled tasks to further enhance the automation experience

PowerShell brings us a productivity boost that will easily repay the time we spend learning how to get the best from the technology. Using PowerShell and WMI together will further enhance our productivity gains as we will see next.

WMI Overview

We will examine the reasons behind using WMI and what it offers us as administrators. WMI has been available to Windows administrators since the days of NT 4. One major point that needs to be remembered is that it isn't a static technology. Each new version of Windows brings changes to the functionality available through WMI—usually by adding extra capabilities but occasionally by removing or radically changing functionality. New versions of applications can have a similar impact; for instance, Exchange 2003 had WMI support but that was removed in Exchange 2007/2010.

WMI AND OFFICE

Microsoft Office 2007 supplied a WMI provider in the shape of the root\MSAPPS12 namespace. This functionality was removed in Office 2010. There will be remnants of the WMI classes left on the system if an upgrade from Office 2007 to 2010 is performed, though they won't be usable.

The only way to be sure that particular functionality is available on your version of Windows is to check the documentation on Microsoft's Developer Network (MSDN). WMI functionality available on a particular system can be discovered in a number of ways using PowerShell and other tools.

There are many examples of automation scripts using WMI. Most of them use VBScript, which gives the unfortunate impression that WMI needs a lot of coding to achieve any gains. This is no longer true. Now we need to consider some of the potential blockers to using WMI. We'll start by finding out what it actually is.

WHAT IS WMI?

Just what is WMI? The abbreviation stands for Windows Management Instrumentation, and the functionality is automatically installed with Windows. The base functionality can be enhanced by adding features and roles to Windows or installing additional applications.

At first glance, it seems like a very large set of stuff that we might be able to use depending on whether we get lucky and someone has mapped out how to use the bit in which we are interested.

At this stage, we need to be aware that WMI doesn't exist in a vacuum. It is Microsoft's implementation of the Common Information Model (CIM), which is produced by the Distributed Management Task Force (DTMF). The CIM (and WMI) defines a series of classes that supply information about our systems and may allow us to directly interact with aspects of local and remote systems.

If we look at the WMI classes available for working with disks by using `Get-WmiObject`:

```
Get-WmiObject -List *disk* | sort name | select name
```

We get the following output:

```
Name
----
CIM_DiskDrive
CIM_DisketteDrive
CIM_DiskPartition
CIM_DiskSpaceCheck
CIM_LogicalDisk
CIM_LogicalDiskBasedOnPartition
CIM_LogicalDiskBasedOnVolumeSet
CIM_RealizesDiskPartition
Win32_DiskDrive
Win32_DiskDrivePhysicalMedia
Win32_DiskDriveToDiskPartition
Win32_DiskPartition
Win32_DiskQuota
Win32_LogicalDisk
Win32_LogicalDiskRootDirectory
Win32_LogicalDiskToPartition
Win32_LogonSessionMappedDisk
Win32_MappedLogicalDisk
Win32_PerfFormattedData_PerfDisk_LogicalDisk
```

```
Win32_PerfFormattedData_PerfDisk_PhysicalDisk
Win32_PerfRawData_PerfDisk_LogicalDisk
Win32_PerfRawData_PerfDisk_PhysicalDisk
```

A number of classes start with "CIM_" and others start with "Win32_". There is not always a one-to-one pairing though major object types such as logical disks are paired. The `CIM_` class is the parent that corresponds to the definition supplied by the DTMF, and the `Win32_` classes are child classes that Microsoft has implemented. In some cases, the classes are identical and, in others, the Microsoft class provides additional functionality.

There are also classes for working with performance counters. We can work with many parts of our systems. Technologies that have this level of power also tend to seem very difficult when we approach them for the first time. WMI is no exception.

WMI IS TOO HARD

Over the years since its introduction, WMI has gained a poor reputation. This has occurred for a number of reasons:

- Inaccessible documentation—many administrators don't think to look on MSDN. I know I didn't when I started using WMI.
- Discovering which classes are available is not always easy.
- Coding WMI can be time consuming.
- A lot of information is held in a coded form; for instance, the `Win32_LogicalDisk` class has a media type property that returns a numeric value. Hard drives are type 3. If that isn't known you can get into problems.
- Class names aren't always consistent; for instance, we have a `Win32_LogicalDisk` class but need to use Win32_DiskDrive rather than a class called `Win32_PhysicalDisk`.

PowerShell itself is also constructed to make WMI usage much easier and more intuitive.

WMI AND POWERSHELL

PowerShell 1.0 has good WMI support. We can use `Get-WmiObject` to access the existing WMI objects, as we saw in listing 1. We also get to create new WMI objects, perform searches, and manipulate the objects we have available.

This capability is raised to a new level with PowerShell 2.0. The PowerShell 1.0 functionality is at least maintained and is often enhanced; for example, we get the capability of working directly with the WMI security levels in the WMI cmdlets. Additional cmdlets are provided to aid modifying and even removing objects. We also get the capability of working directly with WMI events.

Internet code

One thing that we need to consider before progressing too far into WMI and PowerShell code is translating examples from VBScript into PowerShell. This is a necessary skill because a large body of existing examples can make our lives easier. Any script that is downloaded from the Internet or obtained from a book must be tested in your environment to ensure it works as advertised and doesn't have any adverse effects.

I am going to use processes as my example because this will work on everyone's machine and give a sensible result. We could use Get-Process instead of the final code, but I wanted to stick with WMI to make the point. Using VBScript and WMI, we can utilize the code in listing 2 to retrieve some information regarding the processes running on our system. The code is modified from Microsoft's Scripting Guide.

Listing 2 VBScript to retrieve process information

```
set objWMIService = GetObject("winmgmts:" _              #1
      & "{impersonationlevel=impersonate}!\\" _
       & ".\root\cimv2")

set colProcesses = objWMIService.ExecQuery _          #2
      ("SELECT * FROM Win32_Process")

for each objProcess in colProcesses                   #3
    WScript.Echo " "
    WScript.Echo "Process Name : " + objProcess.Name
    WScript.Echo "Handle        : " + objProcess.Handle
    WScript.Echo "Total Handles: " + Cstr(objProcess.HandleCount)
    WScript.Echo "ThreadCount  : " + Cstr(objProcess.ThreadCount)
    WScript.Echo "Path         : " + objProcess.ExecutablePath
next
```

#1 Enables interrogation of WMI
#2 Selects from WMI class
#3 Outputs results

The script starts by creating an object, `objWMIService`, to enable interrogation of the WMI service. (#1) A list of active processes is retrieved by running a WQL query. (#2) The collection of processes is iterated through and we write to screen a caption and the value of a particular property. (#3) This is just a small subset of the available properties to keep the script manageable. Notice that we have to set up the link to WMI, run a query, and then manually define the formatting of our display. All of this takes time even with cut and paste in our editors.

Our code can be directly translated to PowerShell, as shown in listing 3.

Listing 1 PowerShell translation

```
$procs = Get-WmiObject -Query "select * from win32_process"
foreach ($proc in $procs) {
    Write-Host "Name          :" $proc.ProcessName
    Write-Host "Handle        :" $proc.Handle
    Write-Host "Total Handles :" $proc.Handles
    Write-Host "ThreadCount   :" $proc.ThreadCount
    Write-Host "Path          :" $proc.ExecutablePath
}
```

We run the WMI query to select the information we need and put the results into a variable. The variable is a collection of objects representing the different processes. We can then loop through the collection of processes (using the `foreach` command) and for each process in that collection we use the `Write-Host` cmdlet to output a caption and the value of the properties in which we are interested.

VBScript to PowerShell conversions

While working with WMI, it is inevitable that we will end up translating VBScript code into PowerShell given the sheer number of examples that are available from sites such as the Microsoft TechNet Script Center.

The *VBScript to Windows PowerShell conversion guide* is available on the TechNet Script center at http://technet.microsoft.com/en-us/scriptcenter/default.aspx. You will need to search because it does move around on the site.

This should be consulted if it is not obvious how to change a particular piece of VBScript into PowerShell. It also works very well for translating the other way when I needed to produce listing 2.

Using this approach will get the results that we need but it doesn't use PowerShell to its full capabilities. We end up doing the formatting work ourselves rather than leaving it to the machine. Our goal is to get the machines to do as much of the work as possible. We could just run `Get-WmiObject -Class Win32_Process` but this gives us a lot of information to wade through, which is another manual process. We need to select the data we want to see and format it in a sensible way, which leads to the following PowerShell code.

```
Get-WmiObject Win32_process |
Format-Table ProcessName, Handle, Handles,
ThreadCount, ExecutablePath -AutoSize
```

Our final version uses the `Get-WmiObject` cmdlet directly. `Get-WmiObject` will return an object for each process. We use the PowerShell pipeline to pass them directly into a `Format-Table` cmdlet. This combines the data selection and display functionality we have seen earlier and produces a neatly formatted tabular output. If we wanted the output in a list format as we have used in the previous two examples we could substitute `Format-List` for `Format-Table`.

In these simple examples we have progressed from a VBScript that requires a large amount of manual effort to perform the formatting to a PowerShell version that is one line of code that does it automatically for us. The final PowerShell version is small enough that we could type it and run it directly from the command line if required. A better solution is to turn it into a script or function that can accept a machine name as a parameter and then we can use it across our server estate.

Summary

A Windows administrator is under increasing pressure due to the rise in complexity of the environments in which we work and the ever-rising costs of administration. On the one hand, we are being asked to take on more work in terms of a steadily increasing number of servers and a longer list of applications. On the other hand, we are facing demands to cut costs.

The way out of this dilemma is to automate as much of our day to day work as possible. In a Windows environment the pairing of PowerShell and WMI provides an unmatched set of capability:

- PowerShell is available on all currently supported Windows platforms except Windows Server 2008 server core.
- PowerShell support is built into an increasing number of Microsoft and third-party applications.
- WMI provides low-level access to hardware, operating system, and applications enabling full lifecycle management.
- The ability to work with remote systems simplifies administration and stretches the envelope of automation.
- PowerShell enables WMI to be used in a much easier fashion and provides a shortened learning curve to productive use of the technologies.
- There is an existing body of knowledge regarding WMI use that can be readily adapted to use in PowerShell.
- A thriving PowerShell community provides support for PowerShell and WMI usage.
- Both technologies will be available for the foreseeable future, ensuring that our investment continues to show returns.