**Use PowerShell to Quickly Find Installed Software**

http://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-off.png

The Scripting Guys[The Scripting Guys](http://social.technet.microsoft.com/profile/The%20Scripting%20Guys)

The Scripting Guys

Microsoft Corporation

196,050 Points 7 2 3

Recent Achievements

First Helpful Vote Blogs All-Star New Gallery Contributor

[View Profile](http://social.technet.microsoft.com/profile/the%20scripting%20guys/?ws=usercard-hover)

13 Nov 2011 12:01 AM

* Comments [39](http://blogs.technet.com/b/heyscriptingguy/archive/2011/11/13/use-powershell-to-quickly-find-installed-software.aspx#comments)
* Likes

**Summary:** Learn how to use Windows PowerShell to quickly find installed software on local and remote computers.

Microsoft Scripting Guy Ed Wilson here. Guest Blogger Weekend concludes with Marc Carter. The Scripting Wife and I were lucky enough to attend the first PowerShell User Group meeting in Corpus Christi, Texas. It was way cool, and both Marc and his wife Pam are terrific hosts. Here is what Marc has to say about himself.

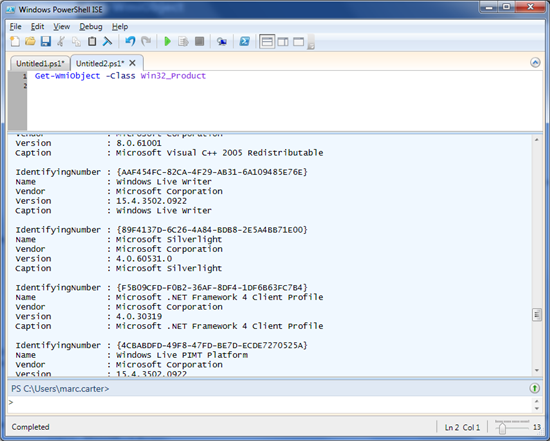
I am currently a senior systems administrator with the Department of the Army. I started in the IT industry in 1996 with DOS and various flavors of \*NIX. I was introduced to VBScript in 2000, and scripting became a regular obsession sometime in 2005. In 2008, I made the move to Windows PowerShell and have never looked back. My daily responsibilities keep me involved with Active Directory, supporting Microsoft Exchange, SharePoint, and various [ASP.NET](http://asp.net/) applications. In 2011, I founded the Corpus Christi PowerShell User Group and try to help bring others up to speed on Windows PowerShell.

Take it away, Marc!

One of the life lessons I have learned over the years working in the IT field as a server administrator is that there are often several different valid responses to a situation. It's one of the things that makes work interesting. Finding the "best" solution to a problem is one of the goals that I think drives many people who are successful at what they do. Occasionally, the best solution is the path of least resistance.

This is one things I love most about working with Windows PowerShell (and scripting in general) is that most problems have more than one solution. Sometimes the "right" way to do something comes down to a matter of opinion or preference. However, sometimes the best solution is dictated by the environment or requirements you are working with.

For instance, let us talk about the task of determining which applications are installed on a system. If you're familiar with the Windows Management Instrumentation (WMI) classes and the wealth of information that can be gathered by utilizing the **Get-WmiObject** cmdlet, an obvious choice might be referencing [the Win32\_product class](http://msdn.microsoft.com/en-us/library/aa394378(v=VS.85).aspx). The **Win32\_Product** represents products as they are installed by Windows Installer. It is a prime example of many of the benefits of WMI. It contains several useful methods and a variety of properties. At first glance, **Win32\_Product** would appear to be one of those best solutions in the path of least resistance scenario. A simple command to query **Win32\_Product** with the associated output is shown in the following image.

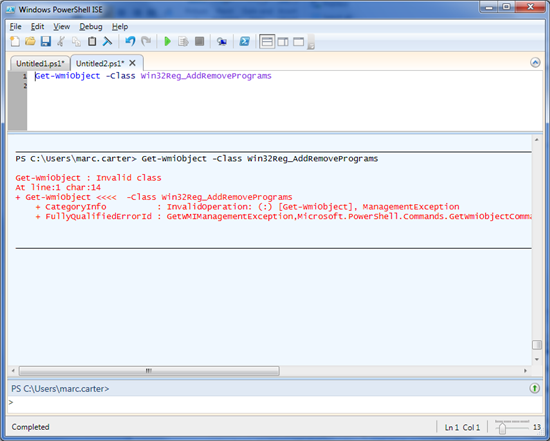
[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/2376.WES_2D00_11_2D00_13_2D00_11_2D00_1.png)

The benefits of this approach are:

* This is a simple and straightforward query: **Get-WmiObject -Class Win32\_Product**.
* It has a high level of detail (for example, Caption, InstallDate, InstallSource, PackageName, Vendor, Version, and so on).

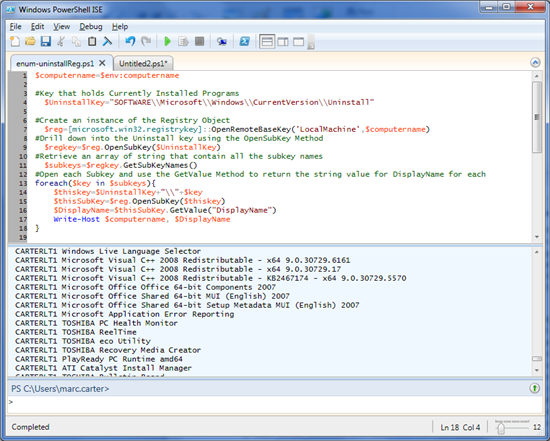
However, because we are talking about alternative routes, let us look at another way to get us to arrive at the same location before we burst the bubble on **Win32\_Product**. Remember, we are simply looking for what has been installed on our systems, and because we have been dealing with WMI, let’s stay with **Get-WmiObject**, but look at a nonstandard class, **Win32Reg\_AddRemovePrograms**.

What is great about **Win32Reg\_AddRemovePrograms** is that it contains similar properties and returns results noticeably quicker than **Win32\_Product**. The command to use this class is shown in the following figure.

[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/7455.WES_2D00_11_2D00_13_2D00_11_2D00_2.png)

Unfortunately, as seen in the preceding figure, **Win32Reg\_AddRemovePrograms** is not a standard Windows class. This WMI class is only loaded during the installation of an SMS/SCCM client. In the example above, running this on my home laptop, you will see the "Invalid class" error if you try querying against it without an SMS/SCCM client installation. It is possible (as Windows PowerShell MVP Marc van Orsouw points out) to add additional keys to WMI using the Registry Provider, and mimic what SMS/SCCM does behind the scenes. Nevertheless, let us save that for another discussion.

One other possibly less obvious and slightly more complicated option is diving into the registry. Obviously, monkeying with the registry is not always an IT pro's first choice because it is sometimes associated with global warming. However, we are just going to query for values and enumerate subkeys. So let's spend a few moments looking at a method of determining which applications are installed courtesy of another Windows PowerShell MVP and Honorary Scripting Guy Sean Kearney (EnergizedTech). In a script that Sean uploaded to the Microsoft TechNet [Script Center Repository](http://gallery.technet.microsoft.com/scriptcenter/e4cdcc2c-185a-43d7-9b44-3de15ba7bf34), Sean references a technique to enumerate through the registry where the "Currently installed programs" list from the Add or Remove Programs tool stores all of the Windows-compatible programs that have an uninstall program. The key referred to is **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall**. The script and associated output are shown in the following figure.

[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/3731.WES_2D00_11_2D00_13_2D00_11_2D00_3.png)

Here are the various registry keys:

#Define the variable to hold the location of Currently Installed Programs  
  $UninstallKey="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall"    
#Create an instance of the Registry Object and open the HKLM base key  
   $reg=[microsoft.win32.registrykey]::OpenRemoteBaseKey('LocalMachine',$computername)    
#Drill down into the Uninstall key using the OpenSubKey Method  
  $regkey=$reg.OpenSubKey($UninstallKey)    
#Retrieve an array of string that contain all the subkey names  
  $subkeys=$regkey.GetSubKeyNames()    
#Open each Subkey and use the GetValue Method to return the string value for DisplayName for each

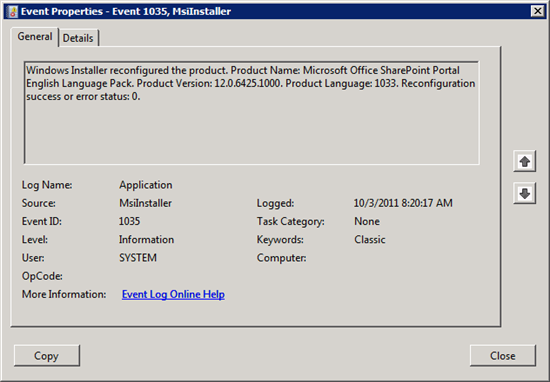
At this point, if you are anything like me, you are probably thinking, "I'll stick with a one-liner and use **Win32\_Product**.” But this brings us back to why we started looking at alternatives in the first place. As it turns out, the action of querying Win32\_Product has the potential to cause [some havoc on your systems](http://support.microsoft.com/kb/974524). Here is the essence of KB974524.

The **Win32\_product** class is not query optimized. Queries such as “select \* from Win32\_Product where (name like 'Sniffer%')” require WMI to use the MSI provider to enumerate all of the installed products and then parse the full list sequentially to handle the “where” clause:,

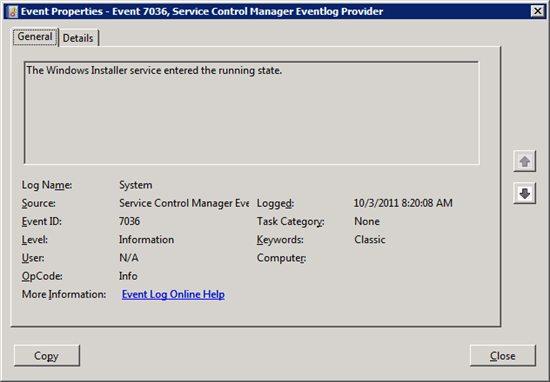
* This process initiates a consistency check of packages installed, and then verifying and repairing the installations.
* If you have an application that makes use of the **Win32\_Product** class, you should contact the vendor to get an updated version that does not use this class.

On Windows Server 2003, Windows Vista, and newer operating systems, querying **Win32\_Product** will trigger Windows Installer to perform a consistency check to verify the health of the application. This consistency check could cause a repair installation to occur. You can confirm this by checking the Windows Application Event log. You will see the following events each time the class is queried and for each product installed:

Event ID: 1035   
Description: Windows Installer reconfigured the product. Product Name: <ProductName>. Product Version: <VersionNumber>. Product Language: <languageID>. Reconfiguration success or error status: 0.

[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/5488.WES_2D00_11_2D00_13_2D00_11_2D00_4.png)

Event ID: 7035/7036  
Description: The Windows Installer service entered the running state.

[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/4048.WES_2D00_11_2D00_13_2D00_11_2D00_5.png)

Windows Installer iterates through each of the installed applications, checks for changes, and takes action accordingly. This would not a terrible thing to do in your dev or test environment. However, I would not recommend querying **Win32\_Product** in your production environment unless you are in a maintenance window.

So what is the best solution to determine installed applications? For me, it is reading from the registry as it involves less risk of invoking changes to our production environment. In addition, because I prefer working with the ISE environment, I have a modified version of Sean’s script that I store in a central location and refer back to whenever I need an updated list of installed applications on our servers. The script points to a CSV file that I keep up to date with a list of servers from our domain.

$computers = Import-Csv "D:\PowerShell\computerlist.csv"

$array = @()

foreach($pc in $computers){

    $computername=$pc.computername

    #Define the variable to hold the location of Currently Installed Programs

    $UninstallKey="SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall"

    #Create an instance of the Registry Object and open the HKLM base key

    $reg=[microsoft.win32.registrykey]::OpenRemoteBaseKey('LocalMachine',$computername)

    #Drill down into the Uninstall key using the OpenSubKey Method

    $regkey=$reg.OpenSubKey($UninstallKey)

    #Retrieve an array of string that contain all the subkey names

    $subkeys=$regkey.GetSubKeyNames()

    #Open each Subkey and use GetValue Method to return the required values for each

    foreach($key in $subkeys){

        $thisKey=$UninstallKey+"\\"+$key

        $thisSubKey=$reg.OpenSubKey($thisKey)

        $obj = New-Object PSObject

        $obj | Add-Member -MemberType NoteProperty -Name "ComputerName" -Value $computername

        $obj | Add-Member -MemberType NoteProperty -Name "DisplayName" -Value $($thisSubKey.GetValue("DisplayName"))

        $obj | Add-Member -MemberType NoteProperty -Name "DisplayVersion" -Value $($thisSubKey.GetValue("DisplayVersion"))

        $obj | Add-Member -MemberType NoteProperty -Name "InstallLocation" -Value $($thisSubKey.GetValue("InstallLocation"))

        $obj | Add-Member -MemberType NoteProperty -Name "Publisher" -Value $($thisSubKey.GetValue("Publisher"))

        $array += $obj

    }

}

$array | Where-Object { $\_.DisplayName } | select ComputerName, DisplayName, DisplayVersion, Publisher | ft -auto

My modified version of Sean’s script creates a **PSObject** to hold the properties I am returning from each registry query, which then get dumped into an array for later use. When I am done, I simply output the array and pass it through a **Where-Object** to display only those entries with something in the **DisplayName**. This is handy because I can then refer back to just the array if I need to supply different output. Say I want to only report on a specific server. I’d change **Where-Object** to something like this:

Where-Object { $\_.DisplayName -and $\_.computerName -eq “thisComputer”}

In conclusion, if you have added Windows PowerShell to your IT tool belt, you have plenty of go-to options when someone asks you, “What’s the best solution to a problem?”

Thank you, Marc, for writing this post and sharing with our readers

I invite you to follow me on [Twitter](http://bit.ly/scriptingguystwitter) and [Facebook](http://bit.ly/scriptingguysfacebook). If you have any questions, send email to me at [scripter@microsoft.com](mailto:scripter@microsoft.com) or post your questions on the [Official Scripting Guys Forum](http://bit.ly/scriptingforum). See you tomorrow. Until then, peace.

**Use PowerShell to Find Installed Software**

http://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.pnghttp://blogs.technet.com/utility/images/star-left-on.pnghttp://blogs.technet.com/utility/images/star-right-on.png

The Scripting Guys[The Scripting Guys](http://social.technet.microsoft.com/profile/The%20Scripting%20Guys)

The Scripting Guys

Microsoft Corporation

196,050 Points 7 2 3

Recent Achievements

First Helpful Vote Blogs All-Star New Gallery Contributor

[View Profile](http://social.technet.microsoft.com/profile/the%20scripting%20guys/?ws=usercard-hover)

15 Nov 2013 12:01 AM

* Comments [15](http://blogs.technet.com/b/heyscriptingguy/archive/2013/11/15/use-powershell-to-find-installed-software.aspx#comments)
* Likes

**Summary**: Guest blogger, Marc Carter, reprises his popular blog post about locating installed software.

Microsoft Scripting Guy, Ed Wilson, is here. [Marc Carter](http://blogs.technet.com/b/heyscriptingguy/archive/tags/marc+carter/) is joining us again today with another guest blog post...

Looking back a couple years ago to my previous post, [Use PowerShell to Quickly Find Installed Software](http://blogs.technet.com/b/heyscriptingguy/archive/2011/11/13/use-powershell-to-quickly-find-installed-software.aspx), I find it interesting to reflect on common issues shared amongst the IT pro community. In our underlying goal to control our environment, whether that environment consists of a desktop computer, a development server, or production data center, we must first discover and understand before we can effectively attempt to control. Such is the case for sys admins when determining what software is currently configuring a server.

But first, let’s have a quick refresher on what initially prompted this discussion…

**Win32\_Product: The Good, the Bad, and the Ugly**

*[Good]* The [Win32\_Product WMI class](http://msdn.microsoft.com/en-us/library/aa394378(v=vs.85).aspx) represents products as they are installed by Windows Installer.

If you choose to query Win32\_Product class by using **Get-WmiObject**, you’ll find yourself *[Bad]* waiting for your query (or application) to return *[Ugly]* a consistency check of packages that are installed as it attempts to verify and repair installs. (For more information, see [Event log message indicates that the Windows Installer reconfigured all installed applications](http://support.microsoft.com/kb/974524)).

**Problem #1: Um, is there a problem, officer?**

Querying the Win32\_Product class to determine installed software is more than likely not your “best” option. Unfortunately, not everyone knows this.

**Solution**: (Understanding) Do your part and help spread the word.

**Problem #2: Identify better alternatives**

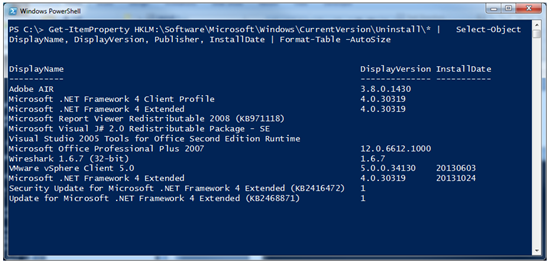
I really like some of the refinements and suggestions within comments that were mentioned by others on my previous post. One of the things I take a lot of pride in is my association with the men and women of US Army and their core values ([The Army Values](http://www.army.mil/values)).

I see that similar mindset and participation reflected in the esprit de corps (or cohesion) of the Windows PowerShell community. As others have pointed out, there are a lot better and easier ways to gather information without invoking the Win32\_Product class. One of my favorite alternatives involved suggestions from Knut Johansen and Mike Crowley: use the PS Registry Provider.

The Windows PowerShell Registry provider lets you get, add, change, clear, and delete registry keys, entries, and values in Windows PowerShell. The Registry provider lets you access a hierarchical namespace that consists of registry keys and subkeys. Registry entries and values are not components of that hierarchy. Instead, they are properties of each of the keys. The Registry provider supports all the cmdlets that contain the “item” noun—that is, the **Item** cmdlets (except **Invoke-Item**) such as **Get-Item**, **Copy-Item**, and **Rename-Item**. Use the **Item** cmdlets when you work with registry keys and subkeys.  
For more information, see [Registry Provider](http://technet.microsoft.com/en-us/library/hh847848.aspx).

In the following example, I use the **Get-ItemProperty** cmdlet to return values from the [Uninstall Registry Key](http://msdn.microsoft.com/en-us/library/aa372105(v=vs.85).aspx) within the HKEY LOCAL MACHINE (HKLM) Registry Provider, selecting specific properties and then formatting output.

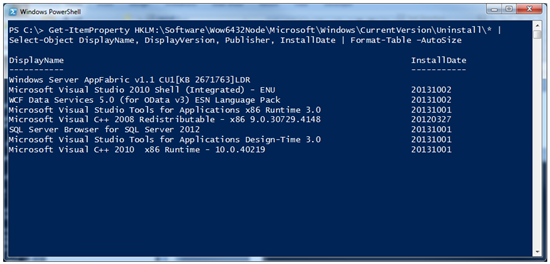
Get-ItemProperty HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\\* |  Select-Object DisplayName, DisplayVersion, Publisher, InstallDate |   
Format-Table –AutoSize

[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/2541.hsg_2D00_11_2D00_15_2D00_13_2D00_1.png)

The [Get-ItemProperty](http://technet.microsoft.com/en-us/library/hh849851.aspx) cmdlet is a great tool because it’s designed to work with data that is exposed by any provider. To get a better idea of the various providers that are available in your session, simply execute the **Get-PSProvider** cmdlet.

And of course, depending on my needs, I could have also used alternative output methods like **Out-GridView** or **Export-Csv**. Either way, we’ve now reduced the process to a one-liner that can be used in 64-bit and 32-bit environments:

Get-ItemProperty **HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\\*** | Select-Object DisplayName, DisplayVersion, Publisher, InstallDate |   
Format-Table –AutoSize

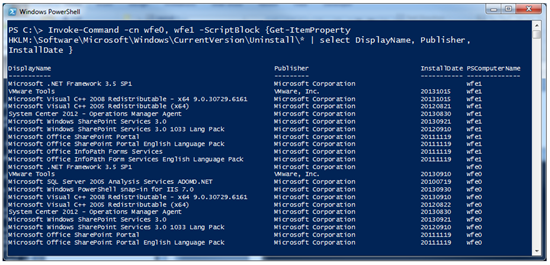
[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/5125.hsg_2D00_11_2D00_15_2D00_13_2D00_2.png)

**Problem #3: Can we make it even more useful?**

Absolutely! We are talking Windows PowerShell after all…

One way that comes to mind (and again, visible within the comments from the previous post), is addressing the issue of how to query multiple remote devices. My solution (or a number of reasons) is to rely on using the **Invoke-Command** cmdlet. In the following example, I query both of my SharePoint Web Front End (WFE) servers by using **Invoke-Command** to execute the same **Get-ItemProperty** on the remote system’s HKLM PS Registry Provider:

Invoke-Command -cn wfe0, wfe1 -ScriptBlock {Get-ItemProperty HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\\* | select DisplayName, Publisher, InstallDate }

[](http://blogs.technet.com/cfs-file.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-00-76-18/5852.hsg_2D00_11_2D00_15_2D00_13_2D00_3.png)

The output now includes the **PSComputerName** column, which will help when I want to sort results down the road. And there we have it…an easy method to report installed software!

I look forward to reading comments from the Windows PowerShell community on other refinements and ways to improve this task. In many ways, I relate our efforts to that of a symphony or band. Each of us plays a different note in that we all hear and see things differently. Put us all together on the same sheet of music, and we have the potential for some awesome melodies.

~Marc

Thank you, Marc, for another awesome blog.

I invite you to follow me on [Twitter](http://bit.ly/scriptingguystwitter) and [Facebook](http://bit.ly/scriptingguysfacebook). If you have any questions, send email to me at [scripter@microsoft.com](mailto:scripter@microsoft.com), or post your questions on the [Official Scripting Guys Forum](http://bit.ly/scriptingforum). See you tomorrow. Until then, peace.

**Ed Wilson, Microsoft Scripting Guy**