June 10th, 2013

Determine Pending Reboot Status—PowerShell Style! Part 1

Doctor Scripto
Scripter

Table of contents

So what's the problem?

The solution journey

The research and validation expedition

Read next

June 10, 2013

PowerTip: Use PowerShell to Create Future Dates

Doctor Scripto

June 11, 2013

Determine Pending Reboot Status—PowerShell Style! Part 2

Doctor Scripto

Summary: Guest blogger, Brian Wilhite, talks about using Windows PowerShell to determine pending reboot status.

Microsoft Scripting Guy, Ed Wilson, is here. Today we have the first of a two-part series about using Windows PowerShell to determine if a reboot is pending. Brian Wilhite, the writer, is no stranger to readers of the Hey, Scripting Guy! Blog—he has written several posts.

Take it away Brian…

So what’s the problem?

Thanks, Ed.

A few months ago one of my coworkers asked if we could determine the pending reboot status of servers in our environment. This is important to know because you will not be able to install updates if the system is pending a reboot from a previous software update. Windows PowerShell is “the” go-to platform to accomplish any task in the Microsoft universe, so I focused my efforts there.

The solution journey

The journey to a solution starts by researching all the avenues a pending reboot is documenting on a system. Furthermore, to be as accurate as possible, I wanted to validate my research by whatever means necessary. After I validate my research, I want to develop reusable code in the form of a function to share with my coworkers and others in the community that may have a similar need. So the birth of Get-PendingReboot began, and you can find it in the TechNet Script Center Repository.

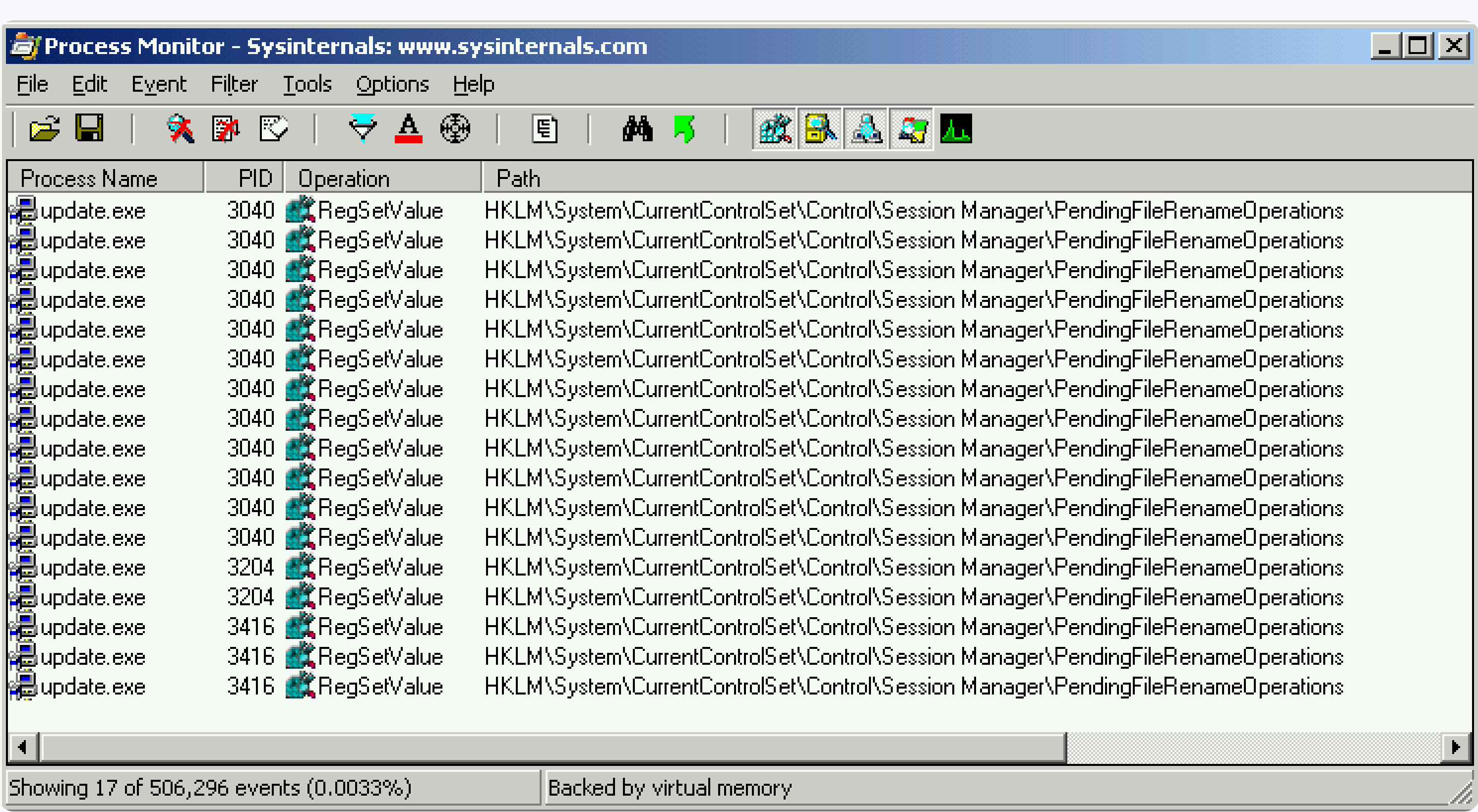
The research and validation expedition

The landscape of our environment, like most, is a mixture of Windows versions ranging from Windows Server 2003 to Windows Server 2012. After searching the web and receiving community feedback, I have compiled several methods that are used to determine a system’s pending reboot status. The methods are (including links for reference):

- Registry: PendingFileRenameOperations
- Registry: WindowsUpdate\Auto Update
- Registry: Component-Based Servicing
- WMI: CCM_ClientUtilities (System Center Configuration Manager clients only)

At this point, my curiosity was piqued, and I wanted to ensure these locations were indeed accurate. I turned to Process Monitor to validate my research. I staged an unpatched computer running Windows Server 2003 with SP2, and I captured events as Windows Update was running and installing updates.

Following is a screenshot of the **RegSetValue** operation for the **PendingFileRenameOperations** multistring value (REG_MULTI_SZ):



I filtered based on **RegSetValue** because this was the operation that actually wrote the value to the registry. Here are the values for this particular round of patching:

An item worthy of note…

In **PendingFileRenameOperations**, the first item in the first pair, "SET21.tmp" is renamed to "wininet.dll" when the server reboots. As you can see, the first line in the pair is the source file and the second line is the destination name. Stated another way, it is moving the file to rename it when the computer reboots, even if the destination file exists. From time-to-time, you'll also observe a pair with a value present in the first line and a blank line immediately afterward. This indicates that the file will be deleted when the computer reboots. When the system performs the Rename and Delete action, the **PendingFileRenameOperations** REG_MULTI_SZ value is cleared.

Next, I performed the same process for **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WindowsUpdate\Auto Update**:

In this screenshot, the **RegCreateKey** operation is called to create the **RebootRequired** registry key, which signifies that a reboot is required.

Next, I captured events on a computer running Windows Server 2008 R2 while it was installing a security update that required a reboot. Notice the **HKLM\Software\Microsoft\Windows\CurrentVersion\Component Based Servicing\RebootPending** key. This key will only exist if the system is pending a reboot. The following image illustrates what occurred with the **RebootPending** key during the security update installation.

Again, you'll notice that the **RegCreateKey** operation calls for the **RebootPending** key.

For the fourth method to work, System Center Configuration Manager 2012 must be installed on the client. When it's installed, the 'ROOT\ccm\ClientSDK' WMI class will be available. This class has a method called **DetermineIfRebootPending**. The method name is a dead give-away as to what it does. The following screenshot illustrates what's returned when calling the method on a client running System Center Configuration Manager 2012. In my example, notice that the method was called, and it returned a False for **RebootPending**. We'll get to the Windows PowerShell code that's used to call this method in my next post.

~Brian

Awesome job, Brian. Way to go, and thanks for contributing to the Windows PowerShell community. Join me tomorrow when Brian will comp his awesome script.

I invite you to follow me on Twitter and Facebook. If you have any questions, send email to me at scripter@microsoft.com, or post your questions on the Official Scripting Guys Forum. See you tomorrow. Until then, peace.

Ed Wilson, Microsoft Scripting Guy