GETTING TO KNOW WMI WITH POWERSHELL

Featuring Goverlan WMIX

Introduction

If you're anywhere in the Microsoft Windows space you've undoubtedly worked with Windows Management Instrumentation (WMI). WMI is one of those features of Windows that many technology professionals work with. Software developers create WMI providers[1] so their applications can interact with WMI. On the flip side, IT pros use WMI and its providers to accomplish a wealth of tasks. WMI can be used to gather information about Windows, the hardware it's running on and the software that's installed. Through its methods, it can change a multitude of settings or even act as a built-in monitor always watching for various activities and triggering actions[2]. WMI is very powerful. However, powerful products are never simple and WMI doesn't let us down there.

When using WMI it's important to understand how it works. You don't have to be a WMI professional but you do need to understand how it's laid out and where to look. If you don't, you'll spend hours sifting through namespaces, classes, methods and properties until your eyes roll back into your head. It can be a miserable experience.

Goverlan WMIX

There have been a few GUI tools come and go over the years but none have been truly easy to use. Some are better than others but overall all of them simply didn't make WMI easy to understand. This is understandable. WMI has a lot of moving parts and it's hard to build an user interface to this beast that doesn't look like something out a Boeing 747's cockpit. However, I think that a tool by GoverLAN called WMIX has finally did it.

WMIX is the best way to explore WMI that I've seen. It has an elegant, modern interface and a "browser" view that allows the user to not just browse WMI namespaces and drill down into classes. It allows you to define "root objects" which can be WMI classes, associations or even queries. This gives you a dashboard to everything that WMI has to offer.

One of the features of the WMIX tool I also like is how much you can not only interact with WMI but also learn about WMI in the process. In this whitepaper, I'll go over some ways in which you can explore WMI using various methods and how WMIX can assist you in that endeavor.

Exploring WMI with WMIX and PowerShell

There are lots of different ways to start exploring WMI. If you're a command-line junkie you could choose PowerShell and use that language's WMI and CIM cmdlets. PowerShell provides all the functionality you need in order to fully explore and utilizes all the capabilities of WMI. However, when you're first starting to learn about what WMI can do it's sometimes easier to start out with a graphical tool. GUIs can provide a more intuitive way for someone that's just looking around and checking out how all of the pieces fit together.

If you prefer to learn via the GUI first, you need a tool that acts as your WMI liaison; a tool that “speaks” WMI and allow you to graphically interact with it more fluidly at least until you can find what you're looking for. After you've been able to explore WMI with a graphical tool, it then might make sense to use PowerShell and begin building scripts to accomplish the task at hand.

In this whitepaper, I will be showing you how to explore WMI using both WMIX and PowerShell.

1 Reading Information

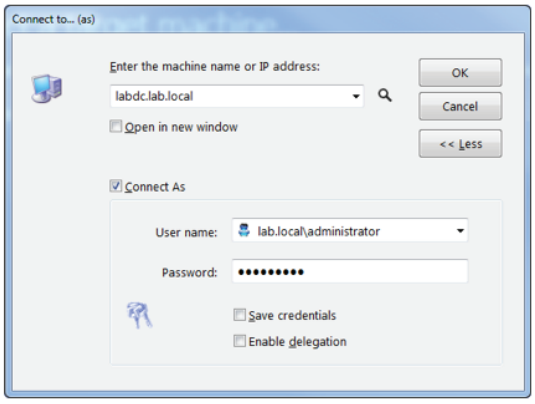
Someone doesn't typically come to work one day and think, "I think I'm going to see what WMI can do." The need to explore WMI typically comes from a problem they've encountered. The time comes when they begin trying to automate various Windows system management tasks. Whether it be as part of a script they're creating or to assist another product gather information on machines, WMI always comes up due to the sheer amount of functionality that can be accomplished with it. WMI is the way to manage a Windows system.

By far, the most common activity to use WMI for is to retrieve information either from a local computer or one or more remote computers. Have you ever needed to find what operating system was installed on a computer, the model of the computer, serial number, free disk space, total memory installed or even what software was installed? You've undoubtedly used WMI.

2 Authentication

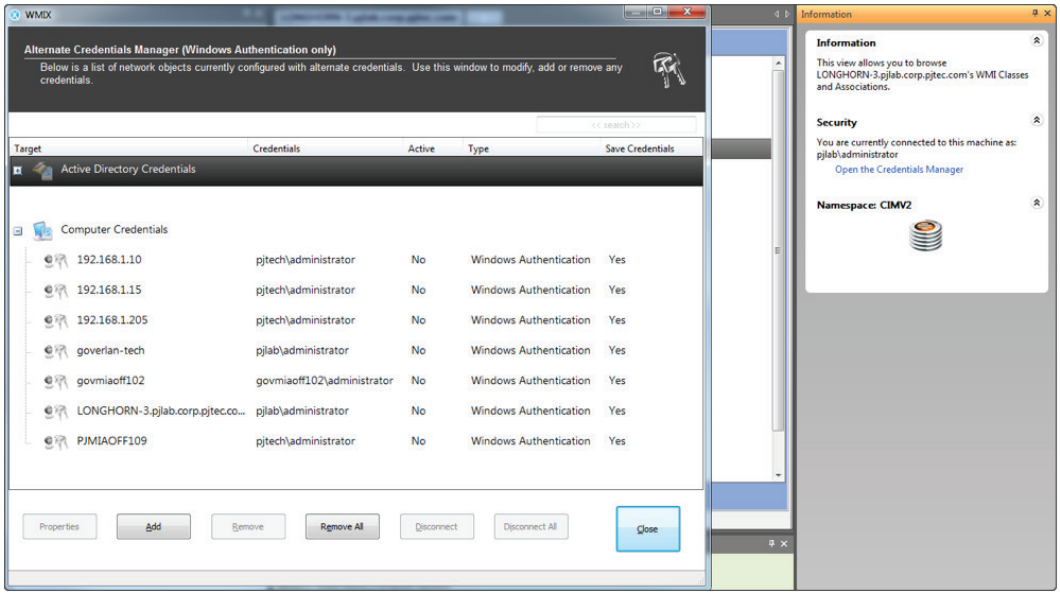
Let's start exploring WMI by trying to find out some information about the operating system a remote computer is running using WMIX. I have a remote computer called labdc in my lab.local domain. Unfortunately, my client is not in that same domain. It's in a workgroup so I can't use the convenience of Active Directory single signon functionality to pass my logged on credentials to the remote computer. Fortunately, GoverLAN gives me the ability to click Connect As and simply provide a username and password I can use.

2 Authentication, continued



The rough equivalent to performing this step with PowerShell would be to first create a Credential object. Once the Credential object has been created, we would then pass that to Get-WmiObject to authenticate as a different user. However, in all truth, there is no way to replicate this step in PowerShell which points out my first key difference in using WMIX over PowerShell to explore WMI. The Get-WmiObject does not simply make a connection to a remote computer. It must have a specific class name or you can use the List parameter as I have below to show all class names in the root\cimv2 name-space. Using a GUI tool allows you make smaller steps.

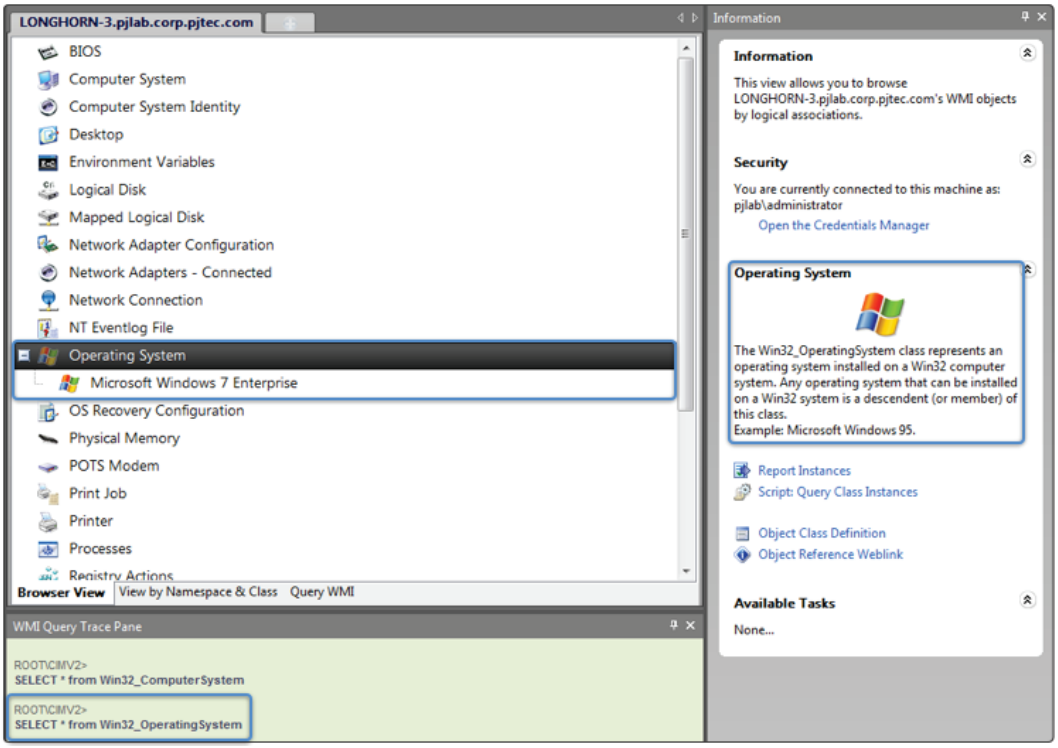
Since I'm using alternate credentials to connect to this remote computer, WMIX shows me at all times I'm using alternate credentials as you can see from the screenshot below. If using PowerShell alone, I would have to include the $Credential variable to Get-WmiObject for every subsequent call I use. By using WMIX, this would be saved for my entire session. Better yet, it even provides an Alternate Credentials Manager tool that allows me to save commonly used credentials.



3 Browsing WMI

Once you've established the connection to the remote computer, we'll then need to look around to figure out where the operating system value is hiding. Using WMIX, which uses a proprietary "browser view" I can quickly scan down all of the WMI classes and immediately see an Operating System category. When double-clicked, it then brought down the operating system name which was exactly what I was looking for.

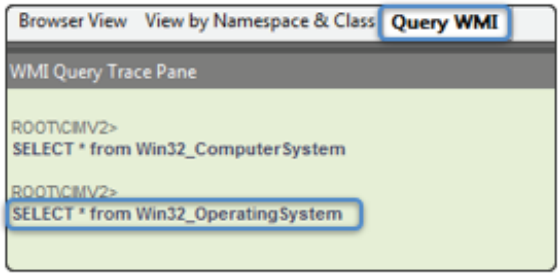
If the name seems to be what I'm looking for I can check out the right-hand side of the screen to view a brief explanation of that that WMI class represents.



Also, you'll notice I highlighted the WMI Query Trace Pane. This is an excellent way to learn about WMI (and eventually using PowerShell to interact with WMI). This pane will show you the exact WQL query that is being ran to display the results you're seeing. You can see that WMIX's browser view is showing you the friendly name "Operating System". However, the actual WMI class name is Win32_OperatingSystem. You'll need to know this once you start writing scripts for WMI. But wait, maybe you don't need to head to a script just yet.

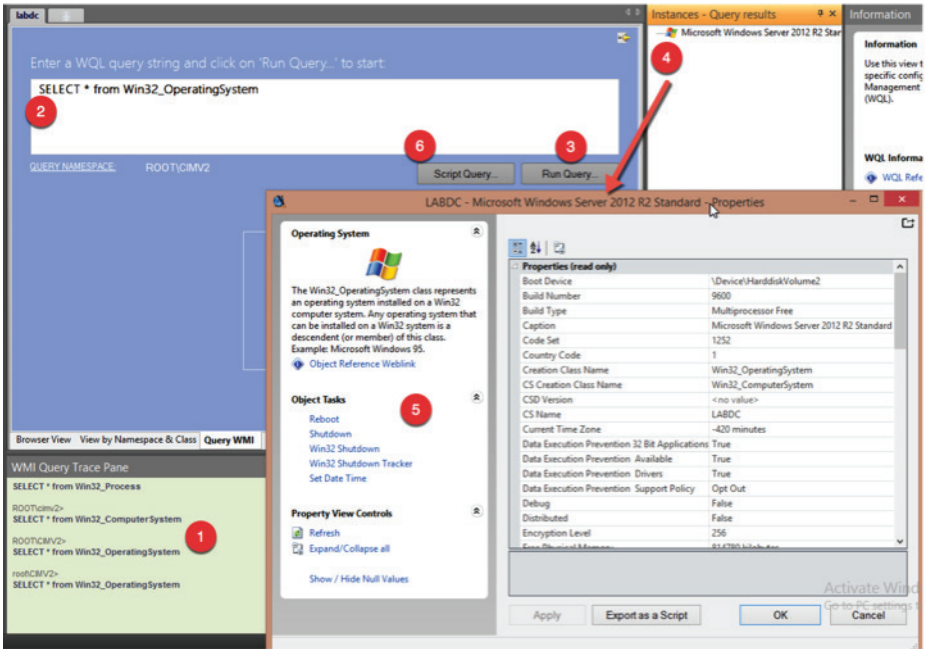
Notice the Query WMI tab at the bottom of the browser view? Let's take that WQL query and click on that tab to check out what other information the Win32_Operat-ingSystem class has to offer.

3 Browsing WMI, continued



As you get to the Query WMI screen, you'll immediately see an option to enter a WQL query string in. In this screenshot below I have copied the contents of #1 over to the #2 box then clicked on Run Query represented by #3. To gather up all the other properties associated with this WMI class, I then double-clicked on the operating system name (#4) in the Instances pane which brought up a ton of information on the Win32_OperatingSystem class.

You can see from this one view that the Win32_OperatingSystem class has a number of read-only properties. This view is also combined with WMI methods that are associated with the Win32_OperatingSystem class (#5). It looks like I can reboot, shutdown or set the date time time this way as well. Nice, added bonus!



3 Browsing WMI, continued

To discover all of this information with PowerShell would look much different. This would not include the description of the class.

Get-WmiObject -ComputerName labdc -Credential $Credential -Class Win32_Operat-
ingSystem | Select-Object *

Get-WmiObject -ComputerName labdc -Credential $Credential -Class Win32_Operat-
ingSystem | get-member –MemberType Method

But, using WMIX, you don't have to know this right off the bat. WMIX has "Export script" buttons hidden everywhere in the tool. In our example above it's represented by #6. This allows you to export a well-written PowerShell script that mimics the behavior of WMIX. This is a great time-saving feature which gives you a pre-built script to work from.

Conclusion

We've covered a typical scenario of exploring WMI with WMIX and PowerShell. By using WMIX as a learning tool, you can see that simply due to the nature of a GUI WMIX allows for easy discoverability. And, due to the extra features like the Alternate Credentials Manager, it will save you time from remembering to include certain parameters on each command you must execute in PowerShell.

WMIX is a great tool to learn and explore WMI. PowerShell is a great tool for management and automation. By leveraging each tool's strengths will allow you to get up to speed on WMI much quicker and to create robust, efficient PowerShell scripts that you can use in your environment today.

References

[1] Developing a WMI Provider – http://bit.ly/1GKyxL5

[2] An Insider's Guide to Using WMI Events and PowerShell – http://bit.ly/1O8q2wh