

Primeros pasos en la escritura de scripts WMI

Ya hemos hablado demasiado, así que ahora pasamos a la acción. PowerShell posee en su juego de commandlets, el commandlet `Get-WmiObject` (alias : `gwmi`). Es gracias a él que vamos a poder dialogar con la capa WMI de nuestro sistema o de un sistema remoto. Verá que es sorprendentemente fácil.

Veamos los parámetros más utilizados (además de los parámetros comunes) del commandlet:

Parámetro	Descripción
Class <String>	Permite definir el nombre de la clase de la que se desea recuperar las instancias.
Property <String>	Permite definir el nombre de la propiedad o conjunto de propiedades a recuperar.
Namespace <String>	Permite definir el espacio de nombres en el que se encuentra la clase. El valor por defecto es <code>root\cimv2</code> .
Query <String>	Permite definir la petición a ejecutar utilizando el lenguaje de petición WQL.
ComputerName <String>	Permite definir el nombre de la máquina en la que se aplica el comando. El valor por defecto es la computadora local (valor «.»).
Filter <String>	Permite definir una cláusula «Where» en formato WQL.
Credential <PSCredential>	Permite proporcionar la información de autenticación si el comando debe realizarse con otra cuenta distinta a la cuenta actual.
List [<SwitchParameter>]	Permite Listar las clases WMI. Esta propiedad trabaja con <code>-namespace</code> . Si se omite el namespace, entonces trabajará en el espacio de nombres <code>root\cimv2</code> .

1. Listar las clases

Hemos visto que existe un cierto número de herramientas que nos permiten encontrar clases en nuestro sistema. Sin embargo, PowerShell también sabe muy bien como hacerlo, quizás de una forma algo menos cómoda que una interfaz gráfica, pero sigue siendo un tema de gustos. Miremos el resultado del siguiente comando:

```
PS > Get-WmiObject -list

...
Name                               Methods                               Properties
----                               -
Win32_CurrentTime                  {}                                    {Day, DayOfWeek, Ho...
Win32_LocalTime                    {}                                    {Day, DayOfWeek, Ho...
Win32_UTCTime                      {}                                    {Day, DayOfWeek, Ho...
Win32_NTLogEvent                   {}                                    {Category, Category...
CIM_ManagedSystemElement           {}                                    {Caption, Descripti...
CIM_LogicalElement                 {}                                    {Caption, Descripti...
CIM_OperatingSystem                {Reboot, Shutdown}                  {Caption, CreationC...
Win32_OperatingSystem              {Reboot, Shutdown...                {BootDevice, BuildN...
CIM_Process                        {}                                    {Caption, CreationC...
Win32_Process                      {Create, Terminat...                {Caption, CommandLi...
CIM_System                         {}                                    {Caption, CreationC...
```

CIM_ComputerSystem	{}	{Caption, CreationC...
CIM_UnitaryComputerSystem	{SetPowerState}	{Caption, CreationC...
Win32_ComputerSystem	{SetPowerState, R...	{AdminPasswordStatu...
CIM_ApplicationSystem	{}	{Caption, CreationC...
Win32_NTDomain	{}	{Caption, ClientSit...
CIM_SoftwareElement	{}	{BuildNumber, Capti...
CIM_BIOSElement	{}	{BuildNumber, Capti...
...		

No mostraremos todos los resultados, pues nos ocuparía páginas y páginas. En efecto, hay un gran número de clases que sería interesante conocer, para ello bastará utilizar este comando:

```
PS > (Get-WmiObject -list).count
965
```

Con Windows Vista, tenemos novecientas sesenta y cinco clases a nuestro alcance. Pero se preguntará «¿pero no faltan?, se nos dijo que había más de mil». Es cierto, pero como habrá podido observar en la explicación sobre el uso del commandlet `Get-WmiObject`, al no haber precisado el espacio de nombres con el parámetro `-namespace, root/cimv2` es el que se utiliza por defecto. Si añadimos las clases de los otros espacios de nombres, tendremos más de 1.000 clases.



El número de clases WMI está en constante crecimiento. No conocemos su número exacto en los diferentes sistemas operativos Microsoft, pero sepa que para cada nueva versión de OS el número de clases aumenta. Esto demuestra, si es que era necesario hacerlo, que WMI es una tecnología muy importante que nos permite efectuar cada vez más tareas.

2. Buscar una clase

Ahora que sabemos listar todas las clases disponibles del sistema, podremos aplicar un filtro sobre el resultado para encontrar la que buscamos. Por ejemplo, supongamos que queremos obtener la fecha y hora de una máquina remota para comprobar si ésta es correcta. De primeras podríamos pensar en una clase cuyo nombre contendría la palabra «date» o «time». Probaremos filtrar por estas palabras:

```
PS > Get-WmiObject -list | Where {$_.name -match 'date'}
```

No ha habido suerte, el filtro por «date» no nos devuelve ninguna clase. Probaremos ahora de filtrar por «time»:

```
PS > Get-WmiObject -list | Where {$_.name -match 'time'}
```



```
NameSpace: ROOT\cimv2
```


Name	Methods	Properties
__TimerNextFiring	{}	{NextEvent64BitTime, TimerId}
MSFT_NetConnectionTimeout	{}	{Milliseconds, SECURITY_DESCRIPTOR, Ser...
MSFT_NetTransactTimeout	{}	{Milliseconds, SECURITY_DESCRIPTOR, Ser...
MSFT_NetReadfileTimeout	{}	{Milliseconds, SECURITY_DESCRIPTOR, TIM...
__TimerEvent	{}	{NumFirings, SECURITY_DESCRIPTOR, TIME...
__TimerInstruction	{}	{SkipIfPassed, TimerId}
__AbsoluteTimerInstruction	{}	{EventDateTime, SkipIfPassed, TimerId}
__IntervalTimerInstruction	{}	{IntervalBetweenEvents, SkipIfPassed, T...
Win32_CurrentTime	{}	{Day, DayOfWeek, Hour, Milliseconds...}
Win32_LocalTime	{}	{Day, DayOfWeek, Hour, Milliseconds...}
Win32_UTCTime	{}	{Day, DayOfWeek, Hour, Milliseconds...}
Win32_TimeZone	{}	{Bias, Caption, DaylightBias, DaylightD...

Ahora sí, hemos obtenido una decena de clases. Nuestro sentido agudo del discernimiento, gracias una cierta experiencia, nos impulsa a listar las instancias de la clase `win32_utctime`:


```
PS > Get-WmiObject Win32_UTCTime
```

```
__GENUS          : 2
__CLASS           : Win32_UTCTime
__SUPERCLASS      : Win32_CurrentTime
__DYNASTY         : Win32_CurrentTime
__RELPATH         : Win32_UTCTime=@
__PROPERTY_COUNT  : 10
__DERIVATION      : {Win32_CurrentTime}
__SERVER         : WIN7_ESCRITORIO
__NAMESPACE       : root\cimv2
__PATH           : \\WIN7_ESCRITORIO\root\cimv2:Win32_UTCTime=@
Day              : 18
DayOfWeek        : 0
Hour             : 9
Milliseconds     : 
Minute           : 47
Month            : 10
Quarter          : 4
Second           : 23
WeekInMonth      : 4
Year             : 2009
```

Hemos logrado determinar en qué clase WMI se «oculta» la hora y la fecha (UTC (*Universal Time Coordinated*)) de nuestro sistema. Únicamente nos falta ahora enviar esta petición a una máquina remota para finalizar nuestro ejemplo. Para ello, será suficiente añadir el parámetro `-computer` a nuestra petición y listo:

```
PS > Get-WmiObject Win32_UTCTime -computer W2K8R2SRV
```

```
__GENUS          : 2
__CLASS           : Win32_UTCTime
__SUPERCLASS      : Win32_CurrentTime
__DYNASTY         : Win32_CurrentTime
__RELPATH         : Win32_UTCTime=@
__PROPERTY_COUNT  : 10
__DERIVATION      : {Win32_CurrentTime}
__SERVER         : W2K8R2SRV
__NAMESPACE       : root\cimv2
__PATH           : \\W2K8R2SRV\root\cimv2:Win32_UTCTime=@
Day              : 18
DayOfWeek        : 0
Hour             : 9
Milliseconds     : 
Minute           : 50
Month            : 10
Quarter          : 4
Second           : 8
WeekInMonth      : 4
Year             : 2009
```

 Para ejecutar una petición WMI a distancia debe ser miembro del grupo Administrador local de la máquina remota o Administrador del dominio.

No obstante podemos también especificar credenciales diferentes durante la ejecución de la petición WMI especificando el parámetro `-credential`, como se muestra a continuación:

```
PS > Get-WmiObject Win32_UTCTime -computer W2K8R2SRV -credential (Get-Credential)
```

Una ventana de diálogo se abrirá entonces y podrá introducir el login y contraseña de administrador de la máquina remota.

3. Buscar una propiedad

A veces sigue siendo difícil encontrar la información que nos interesa, e incluso haciendo una búsqueda lo más precisa posible en el nombre de una clase WMI. En este caso, tendremos que tratar de identificar los nombres de las propiedades que podrían ser pertinentes. En otras palabras, vamos a peinar el nombre de las propiedades de todas las clases WMI de un espacio de nombres.

En este ejemplo, deseáramos obtener el tamaño de la memoria de nuestro ordenador. Como la búsqueda por el nombre de clases no devuelve nada, nos iremos esta vez a la búsqueda de una propiedad cuyo nombre contenga «memory». Para ello, probaremos esta pequeña línea de comandos:

```
PS > Get-WmiObject -namespace root/cimv2 -list -recurse |  
foreach{$_ .PSBase.properties} | where {$_ .name -match 'memory'} |  
Select-Object origin, name -unique
```

Origin	Name
-----	-----
CIM_OperatingSystem	FreePhysicalMemory
CIM_OperatingSystem	FreeVirtualMemory
CIM_OperatingSystem	MaxProcessMemorySize
CIM_OperatingSystem	TotalVirtualMemorySize
CIM_OperatingSystem	TotalVisibleMemorySize
Win32_ComputerSystem	TotalPhysicalMemory
CIM_VideoController	MaxMemorySupported
CIM_VideoController	VideoMemoryType
Win32_DeviceMemoryAddress	MemoryType
CIM_PhysicalMemory	MemoryType
Win32_PhysicalMemoryArray	MemoryDevices
Win32_PhysicalMemoryArray	MemoryErrorCorrection
Win32_NamedJobObjectActgInfo	PeakJobMemoryUsed
Win32_NamedJobObjectActgInfo	PeakProcessMemoryUsed
Win32_WinSAT	MemoryScore
Win32_NamedJobObjectLimitSetting	JobMemoryLimit
Win32_NamedJobObjectLimitSetting	ProcessMemoryLimit
Win32_NetworkAdapterConfiguration	ForwardBufferMemory
CIM_MemoryCheck	MemorySize
CIM_MemoryCapacity	MaximumMemoryCapacity
CIM_MemoryCapacity	MemoryType
CIM_MemoryCapacity	MinimumMemoryCapacity
Win32_PerfFormattedData_Authorizatio...	NumberofScopesloadedinmemory
Win32_PerfRawData_AuthorizationManag...	NumberofScopesloadedinmemory

...

Todo y que este comando tarde un tiempo en ejecutarse, lo cual es completamente normal debido al gran número de clase a analizar, nos devuelve todas las propiedades que cumplen el criterio de nuestra búsqueda. Además, nos muestra el nombre de la clase de la propiedad devuelta, lo que nos facilitará mucho la vida posteriormente como veremos en el siguiente ejemplo.

Observe que hemos tenido que utilizar la propiedad PSBase, que, como recordará, permite pasar entre otras cosas la adaptación de los tipos PowerShell por defecto (véase el Capítulo Control del Shell - Adición de métodos y propiedades personalizadas). De esta manera podemos obtener la propiedad Origin que nos permite saber en qué clase se encuentra la propiedad deseada.

4. Recuperar el tamaño de la memoria física

Ahora que tenemos conocimiento de la clase a utilizar para determinar la cantidad de memoria de un ordenador, hagámos la llamada con la propiedad `TotalPhysicalMemory`.

```
PS > $maquina = Get-WmiObject Win32_ComputerSystem
PS > $maquina.TotalPhysicalMemory
2136428544
```

Podríamos haber escrito también:

```
PS > (Get-WmiObject Win32_ComputerSystem).TotalPhysicalMemory
2136428544
```

Esta formulación, aunque más concisa es un poco menos legible.

El tamaño nos lo devuelve en bytes, para convertirlo en megabytes vamos a dividirlo por 1024×1024 o mejor por el cuantificador de tamaño **1MB**.

```
PS > (Get-WmiObject Win32_ComputerSystem).TotalPhysicalMemory / 1MB
2037,45703125
```

Tenemos entonces 2037 MB de RAM instalados en nuestro sistema, o sea aproximadamente 2 GB de memoria. No tendremos exactamente 2048 MB disponibles ya que estamos utilizando la tarjeta gráfica integrada en la placa base y ésta se reserva algunos MB de RAM para su funcionamiento.

Para redondear el resultado devuelto, podemos recurrir al método estático **round** de la clase **math** del Framework .NET.

```
PS > [math]::round((Get-WmiObject `
Win32_ComputerSystem).TotalPhysicalMemory / 1MB)
2037
```

Para hacer lo mismo en una máquina remota, basta simplemente con añadir el parámetro `-computer` y especificar el nombre de la máquina remota, iese es todo!

```
PS > (Get-WmiObject Win32_ComputerSystem `
-computer maquinaRemota).TotalPhysicalMemory / 1MB
```

Puede constatar la excepcional concisión de nuestros comandos. Los que hayan practicado VBScript anteriormente lo habrán podido observar.

5. Recuperar información del sistema

Seguimos nuestras investigaciones en WMI a la búsqueda de información más general del sistema que la memoria, y

siempre con la clase Win32_ComputerSystem. Veamos la información que nos puede proporcionar:

```
PS > Get-WmiObject Win32_ComputerSystem
```

```
Domain           : WORKGROUP
Manufacturer     : Gigabyte Technology Co., Ltd.
Model            : G33M-DS2R
Name             : WIN7_ESCRITORIO
PrimaryOwnerName : Armando
TotalPhysicalMemory : 2136428544
```

Obtendremos como retorno de este comando una determinada información útil. Esta información no es más que una pequeña muestra de las propiedades de la clase Win32_ComputerSystem. En efecto ésta posee más de cincuenta. Entonces, ¿por qué no las vemos?

Sencillamente no veremos mas datos ya que PowerShell aplica por defecto vistas predefinidas para los objetos WMI y en particular para clase Win32_ComputerSystem. Para verificarlo escriba el comando siguiente:

```
PS > Set-Location $pshome
```

```
PS > Select-String -path *.pslxml -pattern 'win32_computersystem'
```

```
types.pslxml:738:      <Name>System.Management.ManagementObject
                        #root\cimv2\Win32_ComputerSystem</Name>
types.pslxml:840:      <Name>System.Management.ManagementObject
                        #root\cimv2\Win32_ComputerSystemProduct</Name>
```

Este comando nos indica que en el archivo **tipos.ps1xml**, en la línea 738 se encuentra la definición del tipo de la clase Win32_ComputerSystem.

Como en casos anteriores, para utilizar una vista detallada diferente a la de por defecto, podemos escribir el comando siguiente:

```
PS > Get-WmiObject Win32_ComputerSystem | Format-List *
```

```
AdminPasswordStatus      : 3
BootupState              : Normal boot
ChassisBootupState       : 2
KeyboardPasswordStatus   : 3
PowerOnPasswordStatus    : 3
PowerSupplyState         : 2
PowerState               : 0
FrontPanelResetStatus    : 3
ThermalState             : 2
Status                   : OK
Name                     : WIN7_ESCRITORIO
PowerManagementCapabilities :
PowerManagementSupported :
__GENUS                  : 2
__CLASS                   : Win32_ComputerSystem
__SUPERCLASS              : CIM_UnitaryComputerSystem
__DYNASTY                  : CIM_ManagedSystemElement
__RELPATH                 : Win32_ComputerSystem.Name="WIN7_ESCRITORIO"
__PROPERTY_COUNT          : 58
__DERIVATION              : {CIM_UnitaryComputerSystem, CIM_ComputerSystem, CIM_System, CIM_LogicalElement...}
```

```

__SERVER                : WIN7_ESCRITORIO
__NAMESPACE             : root\cimv2
__PATH                  : \\WIN7_ESCRITORIO\root\cimv2:Win32_ComputerSy...
                        Name="WIN7_ESCRITORIO"

AutomaticManagedPagefile : True
AutomaticResetBootOption : True
AutomaticResetCapability : True
BootOptionOnLimit        :
BootOptionOnWatchDog     :
BootROMSupported         : True
Caption                  : WIN7_ESCRITORIO
CreationClassName        : Win32_ComputerSystem
CurrentTimeZone          : 120
DaylightInEffect         : True
Description              : AT/AT COMPATIBLE
DNSHostName              : Win7_Escritorio
Domain                   : WORKGROUP
DomainRole               : 0
EnableDaylightSavingsTime : True
InfraredSupported        : False
InitialLoadInfo          :
InstallDate              :
LastLoadInfo             :
Manufacturer             : Gigabyte Technology Co., Ltd.
Model                    : G33M-DS2R
NameFormat               :
NetworkServerModeEnabled : True
NumberOfLogicalProcessors : 2
NumberOfProcessors       : 1
OEMLogoBitmap           :
OEMStringArray           :
PartOfDomain             : False
PauseAfterReset          : -1
PCSystemType             : 1
PrimaryOwnerContact      :
PrimaryOwnerName         : Armando
ResetCapability          : 1
ResetCount               : -1
ResetLimit               : -1
Roles                    : {LM_Workstation, LM_Server, Print, NT...}
SupportContactDescription :
SystemStartupDelay       :
SystemStartupOptions     :
SystemStartupSetting     :
SystemType               : X86-based PC
TotalPhysicalMemory      : 2136428544
UserName                 : Win7_Escritorio\Armando
WakeUpType               : 6
Workgroup                : WORKGROUP
Scope                    : System.Management.ManagementScope
Path                     : \\WIN7_ESCRITORIO\root\cimv2:Win32_ComputerS...
                        Name="WIN7_ESCRITORIO"

Options                  : System.Management.ObjectGetOptions
ClassPath                : \\WIN7_ESCRITORIO\root\cimv2:Win32_ComputerSy...
Properties                : {AdminPasswordStatus, AutomaticManagedPagef...

```

```

AutomaticResetBootOption,
AutomaticResetCapability, ...}
SystemProperties      : {__GENUS, __CLASS, __SUPERCLASS, __DYNASTY...}
Qualifiers            : {dynamic, Locale, provider, UUID}
Site                  :
Container              :

```

A la vista de este listado, entenderá mejor porqué la vista por defecto se contenta con visualizar una pequeña muestra de las propiedades más significativas.

Acabamos de listar las propiedades y sus valores, veamos ahora cuáles son los métodos de ese objeto. Y para ello, vamos a usar el commandlet `Get-Member -MemberType method`.

```
PS > Get-WmiObject Win32_ComputerSystem | Get-Member -MemberType method
```

```

    TypeName: System.Management.ManagementObject#root\cimv2\
Win32_ComputerSystem

```

Name	MemberType	Definition
----	-----	-----
JoinDomainOrWorkgroup	Method	System.Management.ManagementBaseObj...
Rename	Method	System.Management.ManagementBaseObj...
SetPowerState	Method	System.Management.ManagementBaseObj...
UnjoinDomainOrWorkgroup	Method	System.Management.ManagementBaseObj...

Gracias a `Get-Member` no habrá necesidad de ir explorando el esquema de objetos con una herramienta determinada, ya que PowerShell lo hace por nosotros. Nos devuelve también la definición de cada método para ayudarnos a utilizarlos. Esto es muy bueno para darnos una idea de su utilización, pero cuando se trata de «métodos de riesgo» le aconsejamos que vaya a buscar información más detallada en MSDN.

Podemos observar de paso el `TypeName` de nuestro objeto. Vemos que procede del espacio de nombres `root\cimv2`. Insistimos una vez más, si no se precisa el espacio de nombres, será en éste donde PowerShell irá a buscar todos los objetos.

6. Actuar en el sistema utilizando métodos WMI

Hasta ahora, hemos procurado recuperar información con WMI. En otras palabras no hemos hecho más que emplear las propiedades de objetos. Objetos a los que estamos conectados mediante el commandlet `Get-WmiObject`.

Actuar sobre el sistema es sinónimo de aplicar métodos con objetos WMI. Aunque esto sea posible después de la versión 1 de PowerShell, vamos a ver que la versión 2 simplifica más aún el uso de WMI.

a. Llamada de métodos convencionales

Prestando atención al ejemplo anterior, hemos descubierto cuatro métodos (`JoinDomainOrWorkgroup`, `Rename`, `SetPowerState` y `UnjoinDomainOrWorkgroup`) para actuar sobre nuestro objeto.

```
PS > Get-WmiObject Win32_ComputerSystem | Get-Member -MemberType method
```

```

    TypeName: System.Management.ManagementObject#root\cimv2\
Win32_ComputerSystem

```

Name	MemberType	Definition
----	-----	-----
JoinDomainOrWorkgroup	Method	System.Management.ManagementBaseObj...
Rename	Method	System.Management.ManagementBaseObj...

SetPowerState	Method	System.Management.ManagementBaseObj...
UnjoinDomainOrWorkgroup	Method	System.Management.ManagementBaseObj...

Veamos ahora cómo utilizar uno de estos cuatro métodos. Por ejemplo el que nos permite añadir una máquina a un dominio.

Según la información obtenida de MSDN acerca del funcionamiento del método **JoinDomainOrWorkgroup**, podemos escribir las líneas siguientes para añadir nuestra máquina al dominio *ps-scripting.com*:

```
PS > $maquina = Get-WmiObject Win32_ComputerSystem
PS > $maquina.JoinDomainOrWorkgroup('ps-scripting.com','P@ssw0rd',`
    'administrador@ps-scripting.com',$null,3)

__GENUS          : 2
__CLASS           : __PARAMETERS
__SUPERCLASS     : 
__DYNASTY         : __PARAMETERS
__RELPATH         : 
__PROPERTY_COUNT  : 1
__DERIVATION     : {}
__SERVER         : 
__NAMESPACE      : 
__PATH           : 
ReturnValue       : 0
```

El primer argumento es el nombre completo del dominio objetivo, seguido de la contraseña de una cuenta con los derechos para añadir máquinas al dominio, luego la cuenta en formato *Dominio\Usuario* o *Usuario@Dominio*, seguido de *\$null* para no indicar una unidad organizativa especial de destino, después el valor 3 para especificar que se trata de una adhesión al dominio con creación de la cuenta de ordenador.

Como la operación ha funcionado, obtendremos un código de retorno (**ReturnValue**) cero.

Ahora, no queda más que reiniciar el ordenador para concluir la adhesión en el dominio. Reinicio que podríamos ejecutar localmente con el siguiente comando:

```
PS > $maquina = Get-WmiObject Win32_OperatingSystem
PS > $maquina.Reboot()
```

O en remoto con este otro comando:

```
PS > $maquina = Get-WmiObject Win32_OperatingSystem -computer
NombreDeMquinaRemota
PS > $maquina.Reboot()
```

De nuevo, un valor cero para la propiedad **ReturnValue** significa que la operación ha tenido lugar satisfactoriamente.



En sistemas operativos Windows 7 o Windows Server 2008 R2 es necesario añadir el parámetro `-EnableAllPrivileges` al comando `Get-WMIObject`.

b. Llamada a métodos con **Invoke-WmiMethod**

Le dijimos al inicio de este apartado, que PowerShell v2 simplifica el empleo de los métodos gracias al aporte del commandlet `Invoke-WmiMethod`.

He aquí los parámetros utilizados más comúnmente (excepto parámetros comunes) del commandlet:

Parámetro	Descripción
-----------	-------------

<code>ArgumentList <Object[]></code>	Especifica los parámetros que se van a pasar al método invocado. El valor de este parámetro debe ser una matriz de objetos y estos deben aparecer en el orden requerido por el método invocado.
<code>AsJob [<SwitchParameter>]</code>	Ejecuta el comando como un trabajo en segundo plano. Use este parámetro para ejecutar comandos que tardan mucho tiempo en completarse.
<code>Authentication <AuthenticationLevel></code>	Especifica el nivel de autenticación que se va a utilizar con la conexión de WMI.
<code>Authority <String></code>	Especifica la autoridad que se va a utilizar para autenticar la conexión de WMI. Puede especificar la autenticación estándar NTLM o Kerberos.
<code>Class <String></code>	Especifica la clase de WMI que contiene un método estático al que llamar.
<code>ComputerName <String[]></code>	Especifica el equipo en el que se desea ejecutar la operación de administración. El valor puede ser un nombre de dominio completo, un nombre NetBIOS o una dirección del protocolo Internet (IP).
<code>Credential <PSCredential></code>	Especifica una cuenta de usuario con permiso para realizar esta acción. El valor predeterminado es el usuario actual.
<code>EnableAllPrivileges [<SwitchParameter>]</code>	Habilita todos los privilegios del usuario actual antes de que el comando realice la llamada a WMI.
<code>Impersonation <ImpersonationLevel></code>	Especifica el nivel de suplantación que se va a usar.
<code>InputObject <ManagementObject></code>	Especifica un objeto ManagementObject que se va a usar como entrada. Cuando se utiliza este parámetro, todos los demás parámetros, excepto los parámetros Flag y Argument, se omiten.
<code>Locale <String></code>	Especifica la configuración regional preferida para los objetos de WMI. Especifica el valor del parámetro Locale como una matriz en el formato MS_<LCID> en el orden preferido.
<code>Name <String></code>	Especifica el nombre del método que se va a invocar. Este parámetro es obligatorio y no puede ser NULL ni estar vacío.
<code>Namespace <String></code>	Si se usa con el parámetro Class, este parámetro especifica el espacio de nombres del repositorio de WMI en el que se encuentra el objeto o la clase de WMI a que se hace referencia.
<code>Path <String></code>	Especifica la ruta de acceso al objeto de WMI de una clase de WMI, o bien, la ruta de acceso al objeto de WMI de una instancia de una clase de WMI. La clase o la instancia que especifique debe contener el método que se especifica en el parámetro Name.
<code>ThrottleLimit <Int></code>	Permite al usuario especificar un valor de límite para el número de operaciones de WMI que se pueden ejecutar de manera simultánea. Este parámetro se utiliza con el parámetro AsJob. Este límite se aplica únicamente al comando actual; no se aplica a la sesión ni al equipo.

También podemos realizar un reinicio esta vez con la instrucción Invoke-WmiMethod:

```
PS > $maquina = Get-WmiObject Win32_OperatingSystem -computer
NombreDeMaquinaRemota | Invoke-WmiMethod -name reboot
```

Para que el reinicio sea efectivo, y evitar el error «*The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)*», deberá asegurarse de que el firewall de la máquina remota autoriza las llamadas WMI. Para ello, en la máquina a gestionar, autorice el programa llamado «infraestructura de gestión Windows (WMI)».

Veamos otros pequeños ejemplos de utilización de `Invoke-WmiMethod`, cómo crear un proceso:

```
PS > Invoke-WmiMethod -path Win32_Process -name create -argumentlist  
notepad.exe
```

O cómo añadir una conexión a una impresora de red:

```
PS > Invoke-WmiMethod -path Win32_Printer -name AddPrinterConnection `
-argumentlist \\miServidor\printer1
```

O también definir la impresora «Fax» por defecto:

```
PS > Get-WmiObject -class Win32_Printer -filter "Name='Fax'" |  
Invoke-WmiMethod -name setDefaultPrinter
```

7. Utilización de filtros WMI con WQL

No se hablaría de peticiones WMI si no existiese un lenguaje de peticiones. WMI aporta por tanto el lenguaje WQL (*WMI Query Language*). El WQL es un subconjunto simplificado de SQL ANSI (*Structured Query Language*) muy conocido por los administradores de bases de datos. WQL permite únicamente la recuperación de información; no posee funciones de modificación o de eliminación de datos. No permite ordenar la información devuelta por WMI, pero podremos hacerlo fácilmente gracias a PowerShell y a sus funciones de clasificación nativas (`Sort-Object`).

La utilización de filtros es de gran interés ya que se maneja un volumen importante de datos. Por tanto, al utilizar `Get-WmiObject` *miClase*, nos devolverá todas las instancias de la clase pasada como argumento. Todo y la concisión del comando, éste puede transferir mucha información a través de la red y además requerir mucho tiempo de tratamiento en la parte cliente. Sobre todo si únicamente busca una propiedad de una instancia en particular.

A retener:

El objetivo de los filtros es precisamente efectuar un tratamiento previo por parte del servidor y devolver únicamente la información necesaria al cliente (o consumidor).

`Get-WmiObject` nos permite construir un filtro con los dos parámetros siguientes:

- `-Property`, para recuperar sólo la o las propiedades especificadas.
- `-Query`, para efectuar una petición WQL.

a. Consultar el registro de eventos de una máquina remota

Aunque ya hemos tratado este ejemplo con .NET, podemos también hacerlo utilizando WMI. Esto demuestra que de las tecnologías .NET, COM, ADSI y WMI se solapan, existiendo por tanto numerosas formas de abordar un problema. Sin embargo, el registro de eventos sigue siendo un excelente candidato para la ilustración de nuestras ideas sobre WMI...

Aunque el commandlet `Get-Eventlog` existe en el conjunto estándar de PowerShell, con PowerShell v1 sólo permite manipular los registros de la máquina local. No se pueden recuperar los eventos de una máquina remota.

Para listar los registros de eventos (y no los eventos propiamente dichos) de una máquina remota, tendremos que emplear una petición WMI del tipo:

```
PS > Get-WmiObject Win32_NTEventLogFile -computer 192.168.1.5
```

FileSize	LogfileName	Name	NumberOfRecords
-----	-----	----	-----
131072	Application	C:\WINDOWS\system32\config\	
		AppEv...	394
65536	Directory Service	C:\WINDOWS\system32\config\	
		NTDS...	251
65536	DNS Server	C:\WINDOWS\system32\config\	
		DnsEven...	30
65536	File Replication Service	C:\WINDOWS\system32\config\	
		NtFrs.E...	43
65536	Internet Explorer	C:\WINDOWS\System32\Config\	
		Internet Explo...	0
48168960	Security	C:\WINDOWS\System32\config\	
		SecEvent.Evt	104096
524288	System	C:\WINDOWS\system32\config\	
		SysEvent.Evt	1470
393216	Windows PowerShell	C:\WINDOWS\System32\config\	
		WindowsPowerSh...	475



En este momento, podríamos sin demasiados esfuerzos guardar o borrar un registro. Por ejemplo, tomando el situado en el índice cero de nuestra tabla (recuerde, todas las tablas empiezan por el índice cero), correspondiente al registro de Application.

```
PS > $registro = Get-WmiObject Win32_NTEventLogFile -computer 192.168.1.5
PS > $registro[0].BackupEventlog('c:\backup-Application.evt')
```

¡Atención, la copia se efectúa en la máquina remota!

Para vaciar el registro, basta con sustituir la segunda línea de nuestro ejemplo por:

```
PS > $registro[0].ClearEventlog()
```

Tenga en cuenta también que hay otros muchos métodos posibles, tales como: cambiar los permisos, comprimir, renombrar, etc. Para obtener la lista, como de costumbre, hay que emplear el comando `Get-Member`.

Podemos ver que el registro de eventos Security es muy voluminoso: tiene un tamaño aproximado de 48 Mb. ¡Imagine el tiempo necesario para realizar la consulta si insistimos en traernos en local todo el registro para hacer la búsqueda de un evento en particular!

Para limitar el tiempo de descarga de la información y el tratamiento en la parte cliente, es preferible efectuar una petición WQL para restringir el resultado. Éste tiene la ventaja de ejecutarse desde la máquina remota.

Por ejemplo, queremos saber cuándo han tenido lugar todas las paradas de un servidor remoto. Para ello, buscaremos en el registro de seguridad todos los eventos cuyo código sea 513:

```
PS > Get-WmiObject -query "select eventcode,sourcename,timewritten,
message from win32_NTLogEvent where logfile='Security' AND
EventCode='513'" -computer 192.168.1.5 | Format-List [a-z]*
```

```
EventCode      : 513
Message        : Windows se para.
                Todas las sesiones se cerrarán por esta parada.
```

```
SourceName     : SECURITY
TimeWritten    : 20071019235518.000000+120
```

```
EventCode      : 513
Message        : Windows se para.
                Todas las sesiones se cerrarán por esta parada.

SourceName     : SECURITY
TimeWritten    : 20071019234320.000000+120

...
```

En este ejemplo, ya no hemos utilizado la clase `Win32_NTEventLogFile` sino la clase `Win32_NTLogEvent`. En efecto, esta última contiene todas las instancias correspondientes a los eventos sean cuales sean sus tipos. Por tanto, hemos aplicado un filtro donde le decimos que sólo queremos los eventos contenidos en el registro *Security* Y cuyo código es 513. Observe la utilización de **Format-List [a-z]*** donde pedimos que nos muestre únicamente las propiedades que empiecen por la letra «a» hasta la «z»; es decir, todo el alfabeto excepto caracteres especiales. Hacemos esto ya que sino hubiéramos obtenido entre los resultados nombres de propiedades internas de WMI que comiencen por «__» como «__CLASS» por ejemplo.

b. Examen de los datos en la parte cliente

La otra posibilidad para responder a nuestra necesidad podría ser efectuar una parte del tratamiento de la solicitud por el cliente. Como podrán comprender, es una muy mala idea ya que tendremos que examinar todos los eventos. Y examinar, quiere decir descargar en local todos los eventos. Es decir, en nuestro caso, iaproximadamente 40 MB de datos!

Si lo cree conveniente, puede utilizar esta línea de comandos:

```
PS > Get-WmiObject -query
      'SELECT eventcode,sourcename,timewritten,message,logfile
      FROM win32_NTLogEvent' -computer 192.168.1.5 |
      Where-Object {$_.logfile -eq 'Security' -and $_.eventcode -eq 513} |
      Format-List [a-z]*
```

Aunque en teoría esto debe funcionar para pequeñas cantidades de datos, es posible que obtenga un error de tipo «violación de cuota WMI». Esto puede ocurrir cuando hay demasiada información por recuperar. Sin embargo, el tratamiento en el cliente puede ser muy útil para aquellas personas con pocos conocimientos del lenguaje WQL pero que dominan PowerShell y en particular el commandlet `where-object`. Pero para ello, como habrá visto, es necesario que el volumen de información a manipular sea razonable para maximizar el rendimiento.

8. Ajustes de la seguridad WMI

Hay dos aspectos interesantes cuando se habla de seguridad con WMI. El primero se refiere el acceso remoto a través de firewalls. Dado que WMI se basa en la tecnología COM (en particular los proveedores), en entornos distribuidos es la tecnología DCOM/RPC la que entra en juego. Por lo que habrá que estar muy atento respecto a la configuración del firewall de sus máquinas, ya que DCOM y RPC utilizan puertos que suelen estar filtrados.

El segundo aspecto relativo a la seguridad hace referencia a la cuenta a partir de la cual se efectúan las peticiones WMI; y que estas últimas se aplican localmente o en las máquinas remotas.

Por defecto, `Get-WmiObject` se ejecuta con los permisos del usuario conectado. Normalmente hay que ser Administrador para que las peticiones tengan lugar. La seguridad en nuestros días está cada vez más presente, la prueba de ello es que, desde Windows Vista cuando nos conectamos con una cuenta Administrador, todas nuestras acciones se realizan con un privilegio menor; el de un simple usuario, y será necesario confirmar cualquier acción que queramos efectuar con privilegios más importantes. Este mecanismo se denomina UAC (*User Account Control*).

`Get-WmiObject` sabe liberarse de esta problemática con el parámetro `-credential`. Gracias a éste, será posible especificar

una identidad alternativa para ejecutar una petición WMI.

Ejemplo:

```
PS > $cred = Get-Credential          # se abre una ventana de diálogo
                                     pidiéndole autenticarse

PS > Get-WmiObject Win32_NTEventLogFile -computer 192.168.1.5 -cred $cred
```

Sin entrar en los mecanismos internos y complejos de la seguridad relacionados con WMI, tenga en cuenta también que, desde la versión 2 PowerShell es posible especificar determinados parámetros tales como el nivel de impersonalización de la identidad (impersonation) y el nivel de autenticación (authentication) a utilizar para las conexiones WMI en las máquinas remotas. Estos mecanismos de seguridad son en realidad los de las tecnologías COM y DCOM.

Es sin embargo bastante raro necesitar modificar estos valores. En general nos limitamos a especificar el tándem usuario / contraseña (credentials) administrador de la máquina remota con el parámetro `-credential`.

Veamos una tabla resumen de los diferentes parámetros de seguridad aplicables al grupo de comandos WMI (`Get-WmiObject`, `Set-WmiInstance`, `Invoke-WmiMethod`, `Remove-WmiInstance`) que pueden, en ciertos casos, serle útiles:

Parámetro	Descripción
Authentication <AuthenticationLevel>	Especifica el nivel de autenticación que se va a utilizar con la conexión de WMI. Los valores válidos son: -1: Unchanged 0: Default 1: None (no se realiza ninguna autenticación.) 2: Connect (la autenticación se realiza sólo cuando el cliente establece una relación con la aplicación.) 3: Call (la autenticación se realiza solo al comienzo de cada llamada cuando la aplicación recibe la solicitud.) 4: Packet (se autentican todos los datos que se reciben del cliente.) 5: PacketIntegrity (se autentican y se comprueban todos los datos que se transfieren entre el cliente y la aplicación.) 6: PacketPrivacy (se utilizan las propiedades de los demás niveles de autenticación y se cifran todos los datos.)
Authority <String>	Especifica la autoridad que se va a utilizar para autenticar la conexión de WMI. Puede especificar la autenticación estándar NTLM o Kerberos. Para utilizar NTLM, establezca el valor de autoridad en <code>ntlm:dominio:<nombreDeDominio></code> , donde <code><nombreDeDominio></code> identifica un nombre de dominio NTLM válido. Para utilizar Kerberos, especifique <code>kerberos:<nombreDeDominio\nombreDeServidor></code> . No puede incluir el valor de autoridad cuando se conecta al equipo local.
Credential <PSCredential>	Especifica una cuenta de usuario con permiso para realizar esta acción. El valor predeterminado es el usuario actual. Escriba un nombre de usuario, como "Usuario01", "Dominio01\Usuario01" o Usuario@Contoso.com. O bien, escriba un objeto PSCredential, como el objeto devuelto por el cmdlet <code>Get-Credential</code> . Cuando escriba un nombre de usuario, se le solicitará una contraseña.
EnableAllPrivileges [<SwitchParameter>]	Habilita todos los privilegios del usuario actual antes de que el comando realice la llamada a WMI.
Impersonation	Especifica el nivel de suplantación que se va a usar. Los valores válidos son:

<ImpersonationLevel>	<p>0: Default (lee el Registro local para determinar el nivel de suplantación predeterminado, que suele estar establecido en "3: Impersonate").</p> <p>1: Anonymous (oculta las credenciales del autor de la llamada).</p> <p>2: Identify (permite que los objetos consulten las credenciales del autor de la llamada.)</p> <p>3: Impersonate (permite a los objetos utilizar las credenciales del autor de la llamada.)</p> <p>4: Delegate (permite que los objetos dejen que otros objetos usen las credenciales del autor de la llamada.)</p>
----------------------	--