

Mastering WMI with Windows PowerShell

Jeffery Hicks

Scripting Guru, PowerShell MVP

Pre-requisites for this presentation:

- 1) Windows Administration Experience
- 2) Basic PowerShell

Level: Intermediate

Agenda

- What is WMI?
- Basic WMI Queries
- Credentials and Security
- WMI Testing Tools
- PowerShell and WMI
- Associators Of Queries
- WMI and the Registry
- Asynchronous Events
- All demos will be available at blog.sapien.com



Hands On

- Some time set aside for some hands on work
 - not full blown labs
- Work together
- Open up your copy of the slide deck
- Use books, online resources, cmdlet help, Magic 8 Ball, Ouiji boards or telepathy. Bribery might work too.

What is WMI?

- Windows Management Instrumentation
- Based in industry-standard techniques developed by the Desktop Management Task Force (DMTF)
- Implemented as a service since Windows 2000 (and available for NT 4)
- A system for remotely obtaining management information
- Limited remote configuration

WMI Structure

- Namespaces
- Classes
 - System
 - Product
- Instances

Namespace

- Grouping and organization of similar management objects
- Typically align to products (Windows, SQL, IIS, DNS, etc)
- Default namespace is Root\CIMv2

Classes

- “Live” in namespaces
- Represent manageable components (disks, users, Web sites, NICs, etc)
- Class families will share a prefix like Win32_
- System classes (__GENUS, __PATH etc)

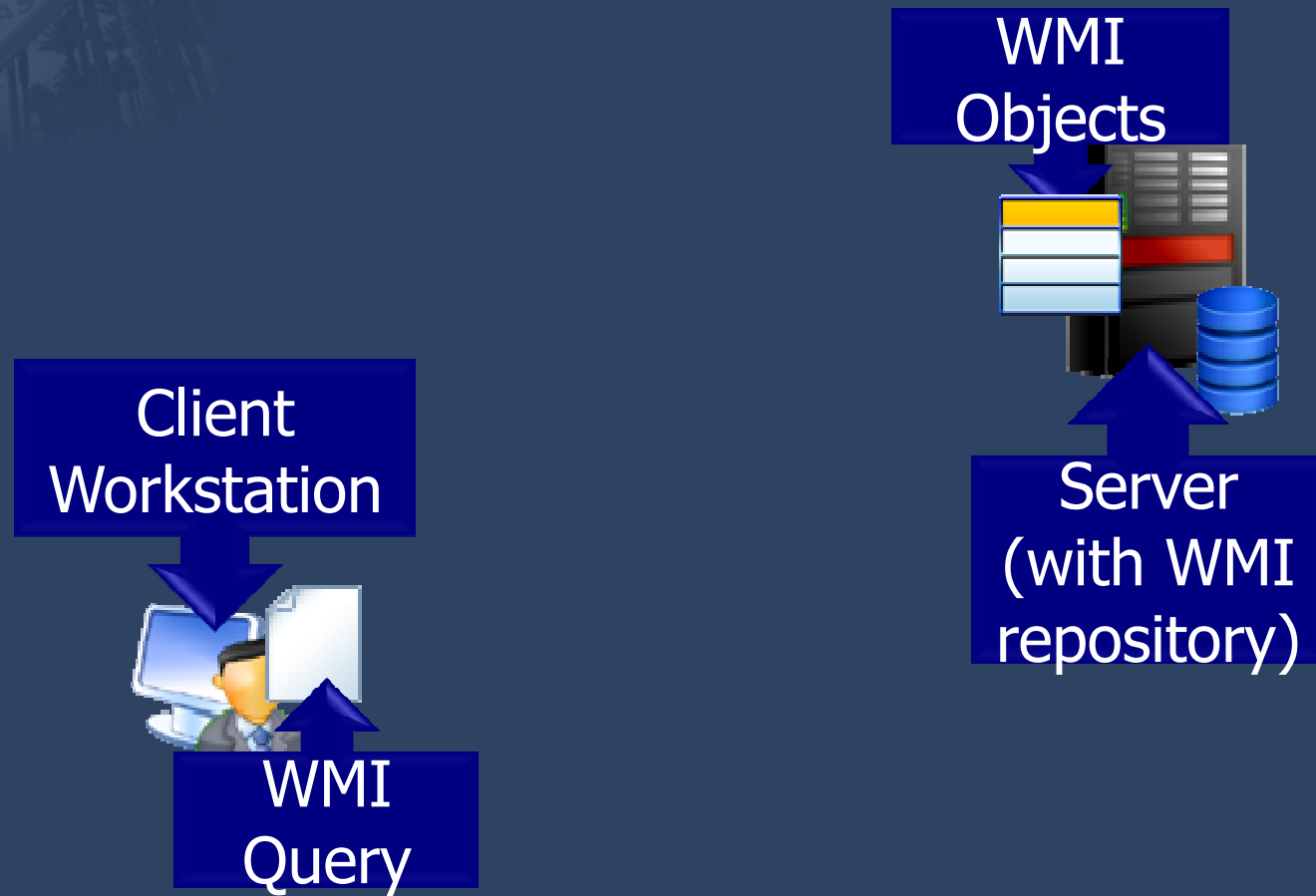
Instances

- Real-world occurrences of a class (two logical disks = two instances of the disk class)
- Instances are *objects*, meaning they have properties
- They can also have *methods*, which are the things you can do with an object (e.g., Disk objects have a ChkDsk method)
- Review properties to see management information; execute methods to make configuration changes

WMI Classes

- Where can you get a list of classes?
- For the core root\cimv2 namespace...
 - Use the WMI Documentation
 - Ask PowerShell for a list of classes

How does WMI work?



WMI Explorers

- Free WMI Explorer from SAPIEN Technologies
- Free WMI Explorer PowerShell script from MoW (the PowerShell Guy)



WMI Queries

- Get all properties for instances of a given class
- Get all objects that match a WMI Query Language (WQL) query
- In PowerShell, the **Get-WMIObject** cmdlet does the work



WQL Queries

- SQL-like syntax
- Specify the properties you want...
- ...the class you're querying...
- ...and the criteria (to filter out instances you don't want)

WQL Example

```
SELECT { * | Property,Property }  
FROM { Class }  
[ WHERE Property = Value ]
```

```
SELECT * FROM Win32_LogicalDisk  
WHERE DriveType = 3
```

```
SELECT DeviceID,Size,Freespace FROM  
Win32_LogicalDisk  
WHERE DeviceID = 'C:'
```

Credentials & Security

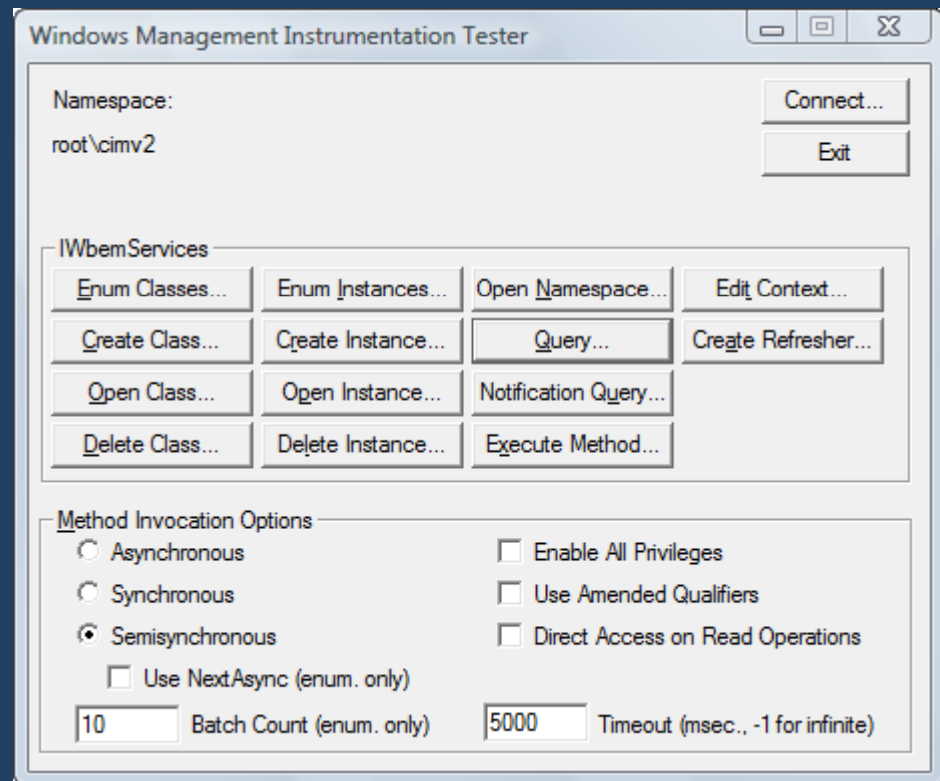
- Uses current credentials by default
- Can specify alternate credentials for remote computers
- Use domain\username (or computername\username) format
- Encrypted over the wire

Credentials & Security

- By default, PowerShell uses PacketPrivacy
- Some tasks require additional privileges even when using admin credentials
- WMI needs RPC ports for remote management

WBEMTest

- Free tool from Microsoft
- Use to verify connectivity
- Test queries



WMI Tools

- Free download from Microsoft
- Internet Explorer based
- CIM Studio
- Object Browser
- Event Registration/Viewer

Hands On

- Using WBEMTEST, get all services on your computer or a remote one if available
- Refine your query and get a list of all stopped services
- Modify your query to show all running services but only getting the name, displayname, its path and service accountname

PowerShell and WMI



Simple WMI

- **Get-WMIObject** *classname*
[-computerName *computername*]
[-credential *PSCredential*]
- Retrieves all instances of the designated class
(optionally, from the specified computer using
the specified credentials)

Get-WMIObject Namespaces

- Default namespace is root\cimv2
- Use –namespace to specify a different one
- Use –list to enumerate all classes of a given namespace

```
PS C:\> Get-WMIObject -list
```

```
PS C:\> Get-WMIObject -namespace  
root\securitycenter
```

Simple WMI

```
PS C:\> Get-WMIObject win32_logicaldisk
```

```
PS C:\> Get-WMIObject win32_bios
```

```
PS C:\> Get-WMIObject win32_computersystem
```

```
PS C:\> Get-WMIObject win32_computersystem  
| select *
```

WQL Queries in PowerShell

```
Get-WMIObject -query "WQL query"
```

```
Gwmi -qu "SELECT * FROM Win32_LogicalDisk  
WHERE DriveType = 3"
```

```
Gwmi -qu "SELECT DeviceID,Size,Freespace FROM  
Win32_LogicalDisk  
WHERE DeviceID = 'C:'"
```


-Filter

- Filter instances using the Where part of a query
- Select * is implied

```
PS C:\> Gwmi -query "Select * from  
win32_logicaldisk where drivetype=3"
```

```
PS C:\> Gwmi win32_logicaldisk -filter  
"drivetype=3"
```

Object Discovery

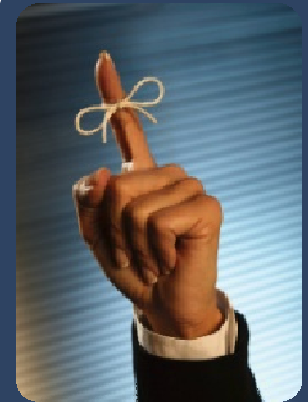
- Pipe a Get-WMIObject expression to Select * to see all the properties, not just the defaults
- Pipe a WMI object to Get-Member to see all of its properties and methods

WMI and the pipeline

- Get-WMIObject returns *objects* with *properties*...
- ...pipe them to Get-Member to see those properties...
- ...or use the objects in the pipeline with Sort, Select, Group, and the other cmdlets you've learned

WMI tip

- A WMI query can potentially return 1 instance, or many instances
- In *all cases*, WMI queries return a *collection* of objects – even if that collection only contains 1 object
- This will be important later – to work with WMI objects in a script, you always have to *enumerate* the collection to get to the individual objects





Working with WMI Objects

- WMI queries return collections
- PowerShell can automatically format results
- Use **ForEach** to enumerate the collection if you need to work with individual instances



WMI Demos

- Queries and reports
- Checking antivirus
- Backing up event logs



Hands on

- List the name, displayname, its path and service accountname of all running services
- Outside of PowerShell, stop a service. Then write a WMI Query to find the service and start it.

Associators Of

- Get related WMI classes
- WIN32_Service
 - Computersystem
 - SystemDriver
 - LoadOrderGroup
 - Antecedent/Precedent services
- Need a specific instance to query

Associators Of

```
$computer = $env:computername
```

```
$Dir="c:\test"
```

```
$Query="ASSOCIATORS OF  
    {win32_directory.name='$Dir'} WHERE  
    ResultClass=CIM_Datafile"
```

```
Get-WmiObject -query $Query -computername  
    $computer | sort FileSize -descending |  
    select Name,FileSize,FileType
```

Demo

- Using WBEMTest
- Get-LocalGroupMembers.ps1
- AssociatorsOfDemo.ps1

Hands On

- List all associated objects with the PowerShell process (\$PID)
- If there's time, select the name of all the associated files

WMI and the Registry

- Does NOT use Get-WMIObject
- Use the StdRegProv to connect to remote systems
- Alternate Credentials more complicated
- Not very quick

StdRegProv

- \$HKLM=2147483650
- \$HKCU=2147483649
- \$HKCR=2147483648
- \$HKEY_USERS=2147483651

```
[WMIClass]$Reg =  
    "\\computename\root\default:StdRegProv"
```



Enumerate Keys

```
$software=$reg.EnumKey($HKLM,"Software")  
$software | foreach {$_.snames}
```

Enumerate Values

```
$regpath="SOFTWARE\Microsoft\Windows\Current  
Version\Run"  
$values=$reg.EnumValues($HKLM,$RegPath)  
$values | foreach {$_.sNames}
```

What About Data?

- You need to call the appropriate method to read the data for a given registry value
 - GetBinaryValue()
 - GetDWORDValue()
 - GetExpandedStringValue()
 - GetMultiStringValue()
 - GetQWORDValue()
 - GetStringValue()

Example

```
$regpath="SOFTWARE\Microsoft\Windows\  
CurrentVersion\Run"
```

```
$value="Windows Defender"
```

```
$reg.GetStringValue($HKLM,$regpath,$value).sValue
```

Discover Data Type

```
for ($i=0;$i -lt $values.snames.count;$i++) {  
    $values.snames[$i]+"="+$values.Types[$i]  
}
```

Windows Defender=2

IgfxTray=1

HotKeysCmds=1

Persistence=1

SigmatelSysTrayApp=1

Types

- 1 = String
- 2 = ExpandedString
- 3 = Binary
- 4 = Dword
- 7 = MultiString



Create a Key

#create a single key

```
$reg.CreateKey($HKCU, "MyStuff")
```

#create a hierarchy

```
$reg.CreateKey($HKCU, "MyStuff\Key1\Key2")
```

Writing Values

- SetBinaryValue()
- SetDWORDValue()
- SetExpandedStringValue()
- SetMultiStringValue()
- SetQWORDValue()
- SetSecurityDescriptor()
- SetStringValue()

Example

#create a string value

```
$reg.SetStringValue($HKCU,"MyStuff\Key1",  
    "SampleKey","I am a string")
```

#create a dword

```
$reg.SetDWORDValue($HKCU,"MyStuff\Key1",  
    "Sample Dword",1024)
```

Delete

#delete a value

```
$reg.DeleteValue($HKCU, "MyStuff\Key1",  
    "SampleKey")
```

#delete a key

```
$reg.Deletekey($HKCU, "MyStuff\Key1\Key2")
```

- You can't delete keys with subkeys

Demo

- Get-PowerShellInstall.ps1

Hands On

- Get the registered owner and organization from:
HKLM\software\microsoft\windows nt\currentversion

```
$HKLM=2147483650
```

```
$computername=$env:computername
```

```
[WMIClass]$Reg =
```

```
"\\$computername\root\default:StdRegProv"
```

Events

- Watching for new event logs
- Watching for file additions
- Watching for process changes
- Watching for service changes

Events

- Use Trace Classes
- Watch for `__Instance` events

Trace Classes

```
get-wmiobject -list | where {$_.name -match  
    "trace"}
```

Win32_SystemTrace

Win32_ProcessTrace

Win32_ProcessStartTrace

Win32_ProcessStopTrace

Win32_ModuleTrace

Win32_ModuleLoadTrace

Win32_ThreadTrace

Win32_ThreadStartTrace

Win32_ThreadStopTrace

Management Object

- System.Management namespace is the .NET class for WMI
- System.Management.ManagementEventWatcher
 - System.Management.ManagementScope
 - **Supports Alternate Credentials**
 - System.Management.WQLEventQuery

```
"Select * from __InstanceCreationEvent Within  
$poll where TargetInstance ISA 'CIM_DATAFILE' AND  
TargetInstance.drive='$drive' AND  
TargetInstance.Path='$folder'"
```

Event Scripts

- Script must stay “alive” to receive events
- PowerShell is “blocked” until the script ends
- Use a realistic and friendly polling interval

Demo

- Get-NewProcessEvent.ps1
- Get-ProcessEvent.ps1

__Instance

- __InstanceCreationEvent
- __InstanceModificationEvent
- __InstanceDeletionEvent

TargetInstance

- TargetInstance ISA *WMIClass*
- __InstanceModificationEvent creates a PreviousInstance object
- Some WMI classes will need additional Where clauses



Demos

- WMI Events using WBEMTest and WMI Tools
- PowerShell scripts

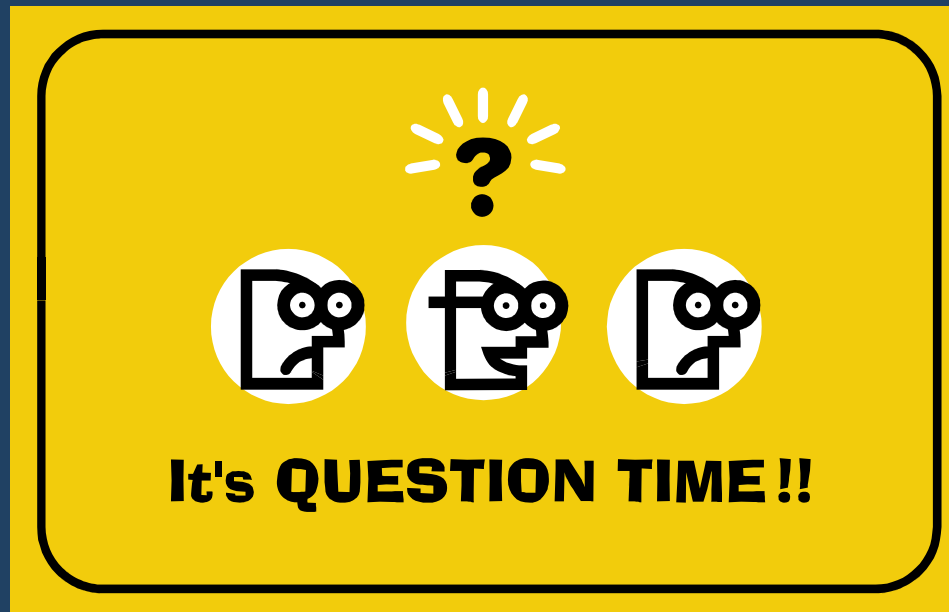
Resources

- Blog.SAPIEN.com
- www.ScriptingAnswers.com
- www.PowerShellCommunity.org
- www.ThePowerShellGuy.com
- www.leeholmes.com/blog

Resources

- Windows PowerShell v1.0: TFM 2nd edition (Jones & Hicks)
- Windows PowerShell Cookbook (Holmes)
- Windows PowerShell in Action (Payette)

Questions



Thank you

- jhicks@sapien.com
- Follow me at twitter.com/jeffhicks

