



realtimepublishers.comtm

*The Administrator
Shortcut Guidetm To*



**VBScripting for
Windows**

Don Jones

Chapter 3: Working with WMI.....42

Classes and Queries42

Scripting and WMI46

 There's No One, Right Way48

 Alternative Credentials49

 Credential Security.....51

What to Do With WMI51

WMI Scriptlets.....54

 Managing Services.....54

 Archive Security Logs57

 Extended WMI.....59

Summary61

Copyright Statement

© 2004 Realtimerepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimerepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimerepublishers.com, Inc or its web site sponsors. In no event shall Realtimerepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimerepublishers.com and the Realtimerepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimerepublishers.com, please contact us via e-mail at info@realtimerepublishers.com.

Chapter 3: Working with WMI

Windows Management Instrumentation (WMI) is the most talked-about technology for managing Windows computers since... well, since TCP/IP, probably. Unfortunately, WMI comes across as the Windows equivalent to quantum physics—it seems complex, nobody's really explaining it clearly, and the documentation requires a few degrees to understand.

The goal of this chapter is to clear some of the fog. It will steer clear from the technically deep, murky world of WMI and stick with the fun and easy-to-understand aspects of WMI. You'll learn enough to use WMI in administrative scripts, which is most likely what you care about most.

Classes and Queries

Everything in WMI starts with *classes*. A class represents some bit of computer hardware, the OS, an application, or something. A class is really an abstract description. For example, there is a class called `Win32_LogicalDisk`, which describes a logical disk. It has properties such as `BlockSize`, `DriveType`, `FreeSpace`, and `FileSystem`, which are all things you would naturally associate with logical disks in Windows.

Each logical disk actually installed under Windows is referred to as an *instance* of the `Win32_LogicalDisk` class. If you have a C drive and a D drive, that is two *instances* of the class. Instances represent actual, living occurrences of the thing that a class describes. Although classes are interesting, you will most often want to deal with the instances. Consider the following example:

1. On a Windows XP machine, select the Start menu, click Run, type
`wbemtest`
and click OK. You should see the dialog box that Figure 3.1 shows.

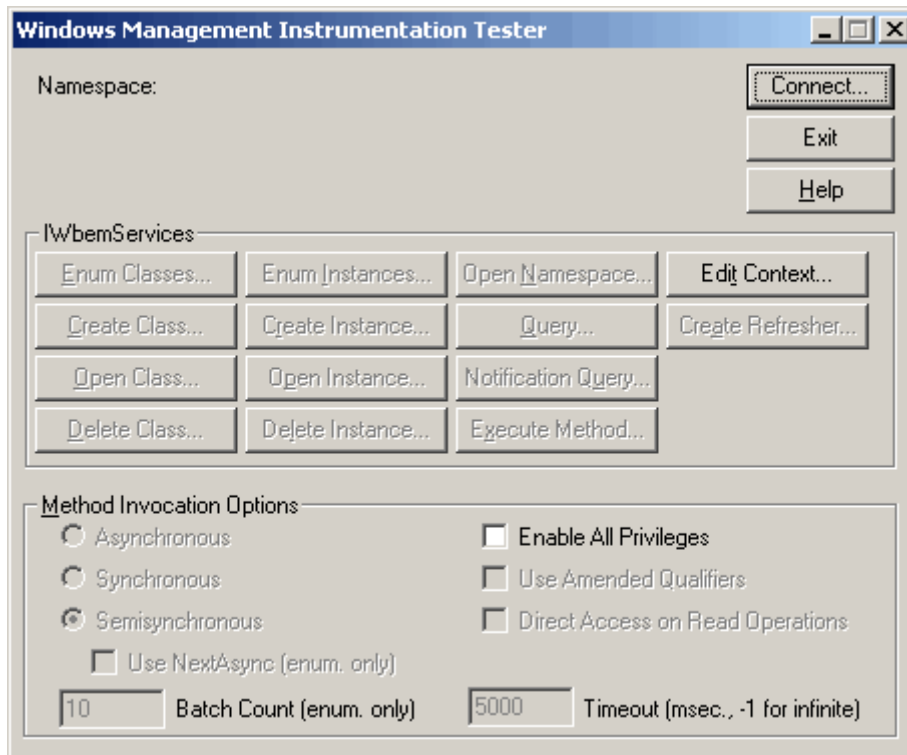


Figure 3.1: The *WBEMTEST* tool.

2. Click Connect. In the Namespace box, type
`root\cimv2`
as Figure 3.2 shows. This namespace is the basic WMI namespace that you'll be using most of the time. Click Connect.

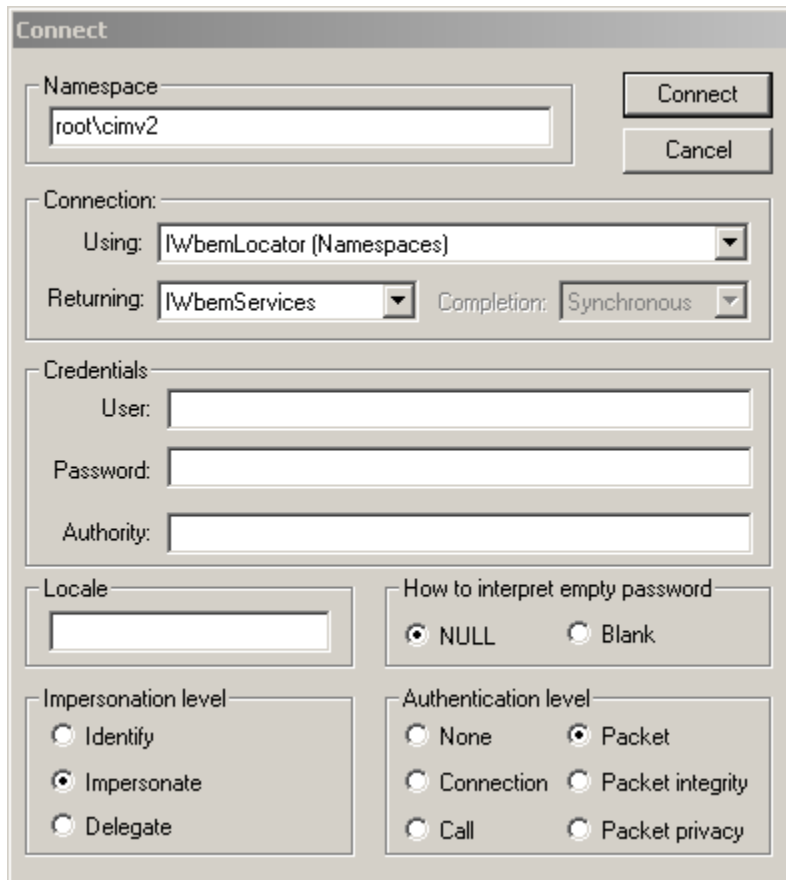


Figure 3.2: Connecting to the WMI namespace.

- Back on the main dialog box, click Query. Select WQL for the query type, and enter `SELECT * FROM Win32_LogicalDisk` for the actual query, as Figure 3.3 shows. Click Apply.

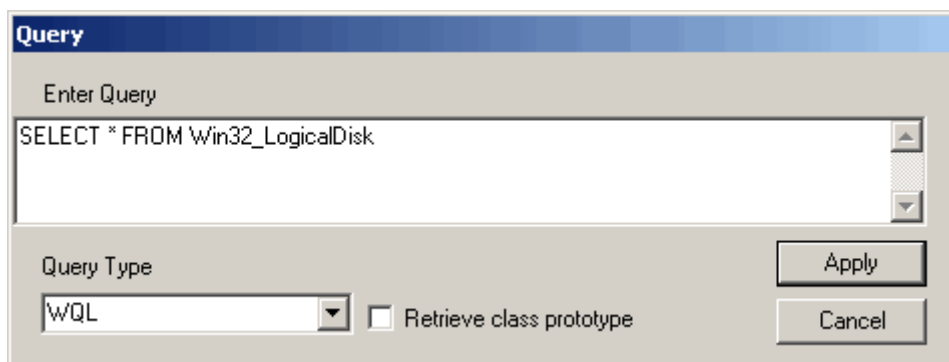


Figure 3.3: Entering a WMI query.

- You should get something like what Figure 3.4 shows, listing each instance that your query returned.

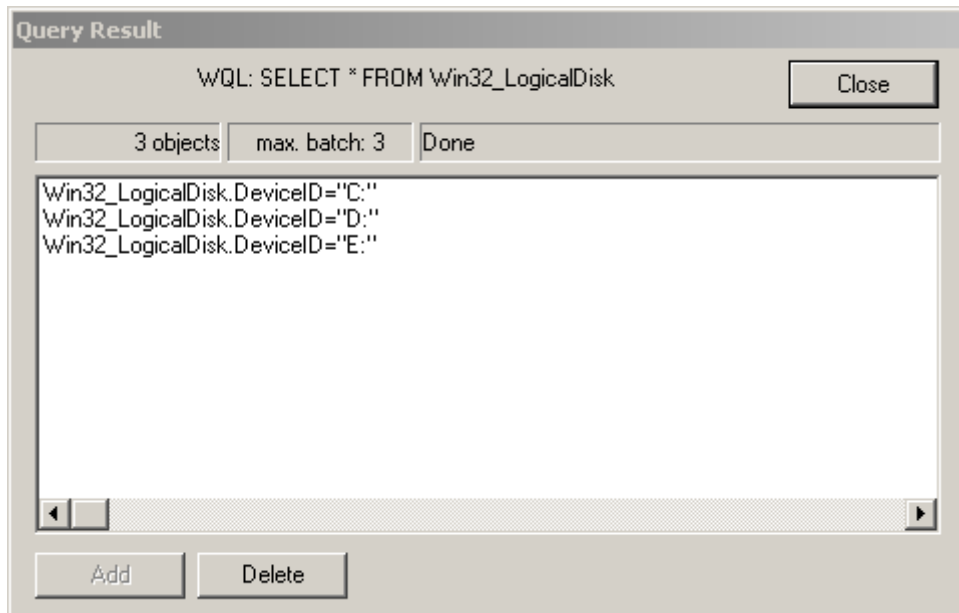


Figure 3.4: Reviewing instances returned by the query.

5. Double-click an instance, and you'll see its properties, as Figure 3.5 shows. Notice the useful information such as FileSystem and FreeSpace.

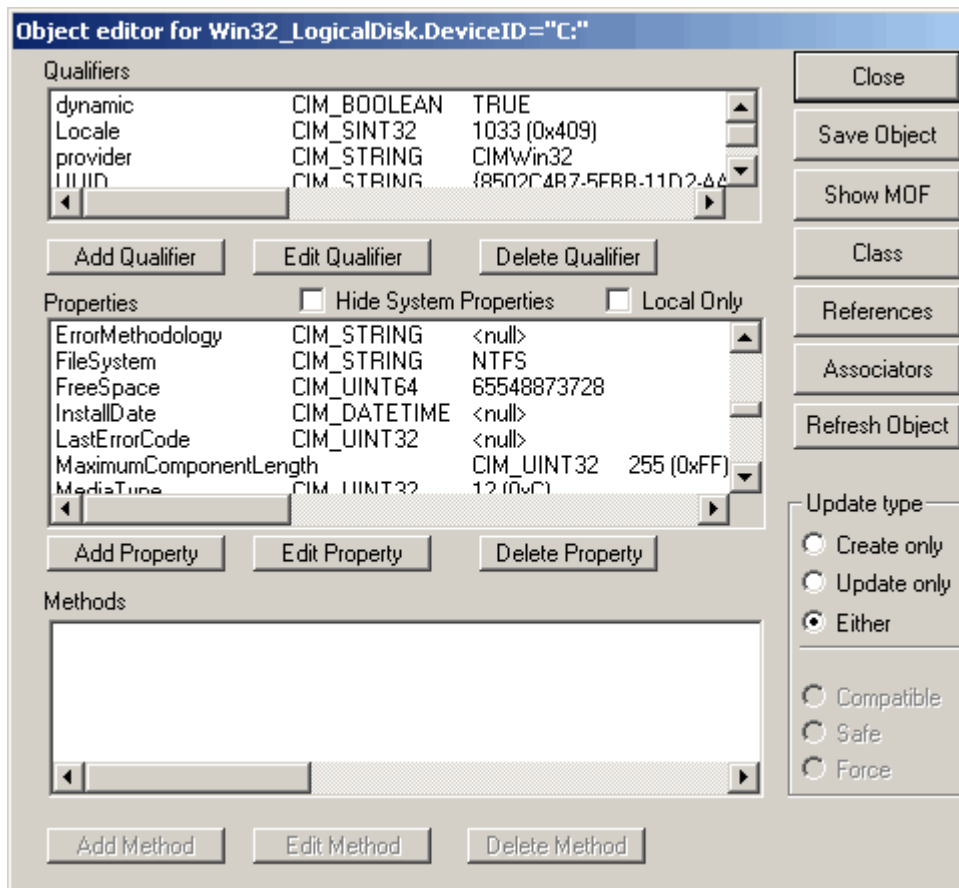



Figure 3.5: Reviewing an instance's properties.

Shazam, you're a WMI guru! Well, almost. This example is meant to illustrate the relationship of classes, instances, and properties, and how they're used to deliver information about your computer.

 The WBEMTEST tool is built right into Windows XP and WS2K3. The Web-Based Enterprise Management (WBEM) is the standard that WMI implements. Common Information Model (CIM—as in cimv2) is a way of standardizing class and property naming and relationships. The CIM was created by the Desktop Management Task Force (DMTF), an industry task force that includes Microsoft.

Scripting and WMI

Although running queries with WBEMTEST is a great way to test WMI queries, there are more useful WMI administrative applications. As an illustration, consider the Win23_LogicalDisk class script that Listing 3.1 shows.

```
On Error Resume Next
Dim strComputer
Dim objWMIService
Dim propValue
Dim colItems

strComputer = "."
Set objWMIService = GetObject("winmgmts:\\\" & _
    strComputer & "\\root\cimv2")
Set colItems = objWMIService.ExecQuery("Select * from \" & _
    \"Win32_LogicalDisk\",,48)
For Each objItem in colItems
    WScript.Echo "Access: " & objItem.Access
    WScript.Echo "Availability: " & objItem.Availability
    WScript.Echo "BlockSize: " & objItem.BlockSize
    WScript.Echo "Caption: " & objItem.Caption
    WScript.Echo "Compressed: " & objItem.Compressed
    WScript.Echo "ConfigManagerErrorCode: " & _
        objItem.ConfigManagerErrorCode
    WScript.Echo "ConfigManagerUserConfig: " & _
        objItem.ConfigManagerUserConfig
    WScript.Echo "CreationClassName: " & objItem.CreationClassName
    WScript.Echo "Description: " & objItem.Description
    WScript.Echo "DeviceID: " & objItem.DeviceID
    WScript.Echo "DriveType: " & objItem.DriveType
    WScript.Echo "ErrorCleared: " & objItem.ErrorCleared
    WScript.Echo "ErrorDescription: " & objItem.ErrorDescription
    WScript.Echo "ErrorMethodology: " & objItem.ErrorMethodology
    WScript.Echo "FileSystem: " & objItem.FileSystem
    WScript.Echo "FreeSpace: " & objItem.FreeSpace
    WScript.Echo "InstallDate: " & objItem.InstallDate
    WScript.Echo "LastErrorCode: " & objItem.LastErrorCode
    WScript.Echo "MaximumComponentLength: " & _
        objItem.MaximumComponentLength
    WScript.Echo "MediaType: " & objItem.MediaType
    WScript.Echo "Name: " & objItem.Name
    WScript.Echo "NumberOfBlocks: " & objItem.NumberOfBlocks
    WScript.Echo "PNPDeviceID: " & objItem.PNPDeviceID

    for each propValue in objItem.PowerManagementCapabilities
```



```

WScript.Echo "PowerManagementCapabilities: " & _
propValue
next

WScript.Echo "PowerManagementSupported: " & _
objItem.PowerManagementSupported
WScript.Echo "ProviderName: " & objItem.ProviderName
WScript.Echo "Purpose: " & objItem.Purpose
WScript.Echo "QuotasDisabled: " & objItem.QuotasDisabled
WScript.Echo "QuotasIncomplete: " & objItem.QuotasIncomplete
WScript.Echo "QuotasRebuilding: " & objItem.QuotasRebuilding
WScript.Echo "Size: " & objItem.Size
WScript.Echo "Status: " & objItem.Status
WScript.Echo "StatusInfo: " & objItem.StatusInfo
WScript.Echo "SupportsDiskQuotas: " & objItem.SupportsDiskQuotas
WScript.Echo "SupportsFileBasedCompression: " & _
objItem.SupportsFileBasedCompression
WScript.Echo "SystemCreationClassName: " & _
objItem.SystemCreationClassName
WScript.Echo "SystemName: " & objItem.SystemName
WScript.Echo "VolumeDirty: " & objItem.VolumeDirty
WScript.Echo "VolumeName: " & objItem.VolumeName
WScript.Echo "VolumeSerialNumber: " & objItem.VolumeSerialNumber
Next

```

Listing 3.1: Example WMI script that involves the Win23_LogicalDisk.



The script that Listing 3.1 shows was produced by the WMI Wizard of the PrimalScript script editor (see <http://www.primalscript.com> for details). The helpful Scripting Guys at Microsoft provide a free WMI Scriptomatic that is similar to the WMI Wizard in PrimalScript; you can download this Scriptomatic from <http://www.microsoft.com/technet/community/scriptcenter/tools/wmimatic.msp>.

The script in Listing 3.1 begins by declaring a few variables. The meat of the script is only a few lines long:

```


strComputer = "."
Set objWMIService = GetObject("winmgmts:\\." & _
strComputer & "\root\cimv2")
Set colItems = objWMIService.ExecQuery("Select * from " & _
"Win32_LogicalDisk", , 48)

```

These lines set a variable named `strComputer` equal to `."`, which happens to be the name of the local computer. You could set this variable to another computer name to pull this information from a remote machine.

Next, the script uses the `Set` keyword and the `GetObject()` function to retrieve an object reference and assign it to the variable `objWMIService`, which is exactly how ADSI works (as you remember from the previous chapter). The object being retrieved is the WMI Service, a background service running on all Win2K and later machines (and on NT machines on which WMI has been installed). The computer name as well as the namespace to connect to—`root\cimv2`—are passed as part of the `GetObject()` method.

Finally, the script uses the `ExecQuery` method to ask the WMI Service to execute a query. The example query should look familiar: it is the same one used in the `WBEMTEST` example earlier. The variable `colItems`, then, will contain a collection of instances, just as Figure 3.4 displays a collection of instances. The last part of the script uses a `For Each...Next` loop to go through each item (instance) in the collection, displaying its properties.

 By the way, running this script under `CScript.exe`, rather than `WScript.exe`, is more efficient. The reason is that the `WScript.Echo` statements, under `WScript`, produce a *lot* of message boxes, which require you to click OK in order to continue. `CScript` interprets `WScript.Echo` as simple command-line output, which doesn't require interaction in order to continue.

There's No One, Right Way

There are many ways to connect to WMI. The `GetObject()` method, used with the `winmgmts://moniker`, as it's called, is a popular way and you'll see it in a lot of examples. One of the reasons it's popular is that it makes it relatively easy to play funky games with security.

By default, a WMI query such as the previous example script connects to the WMI Service, which runs under the `LocalSystem` account. Thus, queries get executed as `LocalSystem`, which might not always be what you want (because, for example, `LocalSystem` isn't allowed to perform a direct shutdown of a computer).

Suppose you're logged on to `ComputerA` as a Domain Admin, and you want to shut down `ComputerB`, which is a domain member. You're an administrator on `ComputerB`, so you're allowed to do so; what you need WMI to do is *impersonate* your credentials for the shutdown task. The `winmgmts://moniker` provides this capability. Listing 3.2 provides an example.


```
strComputer = "computerb"

Set objWMIService = GetObject("winmgmts:" & _
    "{impersonationLevel=impersonate,(Shutdown)}!\\\" & _
    strComputer & "\root\cimv2")

Set colOperatingSystems = objWMIService.ExecQuery _
    ("Select * from Win32_OperatingSystem")

For Each objOperatingSystem in colOperatingSystems
    ObjOperatingSystem.Reboot()
Next
```

Listing 3.2: An example script that uses the `winmgmts://moniker`.

 If you read this script carefully, you'll notice that it appears to be issuing a `Reboot()` command to multiple instances of `Win32_OperatingSystem`, as if one computer could actually contain multiple running instances of Windows. Today, computers can't, but someday they might; WMI is built to recognize the existence of multiple active OSs; thus, querying `Win32_OperatingSystem` could, in theory, return multiple instances.

Notice that some extra text has been tacked into the winmgmts:\\ connection. The complete query is

```
winmgmts:{impersonationLevel=impersonate,(Shutdown)}!\\computerb\
root\cimv2
```

This query tells WMI to impersonate you for the task of shutting down the computer; WMI will try to acquire the necessary permissions on the remote machine and will return an error if it can't.

Alternative Credentials

What the winmgmts://: moniker isn't so good at is providing alternative credentials; meaning ones you're not currently logged on with. In other words, suppose you're logged onto ComputerA as a mere mortal, and want to perform a Domain Admin-style action on ComputerB. The winmgmts:// moniker can't help, but there is another way.

The solution is to fire up a local DLL on the machine running the script. The DLL in question is the WbemScripting.SWbemLocator object, which has the special powers necessary to create a connection to a remote WMI Service and pass alternative credentials. Listing 3.3 provides an example of how to do so.

```
Dim oLocator, oService
Set oLocator = CreateObject("WbemScripting.SWbemLocator")
Set oService = oLocator.ConnectServer( _
    "computerb", "root\cimv2", "Administrator", "Password!")
oService.Security_.impersonationlevel = 3
oService.Security_.Privileges.Add 18
oService.Security_.Privileges.Add 23

Dim oItem
For Each oItem in oService.InstancesOf("Win32_OperatingSystem")
    oItem.Reboot()
Next
```

Listing 3.3: Example script that uses the WbemScripting.SWbemLocator object.

This script does pretty much the same thing that the previous script does, except that this script allows you to pass in alternative credentials. The Locator has a ConnectServer() method, which accepts the server name, namespace, and credentials you want to use. The method returns a reference to the remote WMI Service, which this script references with the variable oService.

The script also messes around with the oddly named Security_ property, which is actually the SWbemSecurity object. One of its properties is impersonationlevel, which is how the WMI Service is told to set an impersonation level. In other words, how should the service impersonate the credentials you've passed along? There are a few values you can provide here:

- 1 means Anonymous, so the credentials you pass in are hidden—that is, not revealed to WMI. WMI probably won't be able to perform whatever you wanted it to with this value.
- 2 means Identify, meaning objects can query the credentials you pass in but not use them. WMI will probably also not work at this level.

- 3 means Impersonate, meaning objects can utilize the credentials you pass in. This value is what WMI usually needs.
- 4 means Delegate, which on Win2K and later allows objects to allow *other* objects to use the credentials you pass in. This value will work with WMI but is usually overkill and may be a security risk.

The script then adds some privileges to the Security_ property's Privileges object. Possible values are:

- 3—Lock physical pages in memory
- 4—Increase the quota assigned to a process
- 5—Create a machine account
- 6—Act as part of the trusted computer base
- 7—Control and view audit messages; required for all tasks reserved for security operators
- 8—Take ownership of an object
- 9—Load or unload a device driver
- 10—Gather profiling information (performance counters) for the system
- 11—Modify system time
- 12—Gather profiling information for a single process
- 13—Increase base priority for a process
- 14—Create a paging file
- 15—Create a permanent object
- 16—Perform backup operations
- 17—Perform restore operations
- 18—Shut down the local system
- 19—Debug a process
- 20—Generate audit log entries
- 23—Shut down a system remotely
- 24—Remove computer from a docking station
- 25—Synchronize directory service data
- 26—Enable computer and user accounts for delegation

This script added both the local and remote shutdown permissions, allowing it to shut down either the local system or a remote system.

Credential Security

Hardcoding usernames and passwords into a script is a horrible, horrible idea. Scripts cannot be reliably protected so that the credentials won't be divulged to someone who shouldn't have them. An alternative is to have the script prompt for credentials using `InputBox()`.

Microsoft provides a free Script Encoder, which is intended to hide the contents of your script from prying eyes while still allowing the script to execute. *This tool is not reliable enough to protect hardcoded credentials.* The encryption keys used by Script Encoder are well-known, and several easily obtainable script decoders exist. Do not imagine for a moment that obscuring credentials with the Script Encoder will protect them; "security through obscurity" is no security at all.

What to Do With WMI

As everything in WMI is based on classes, what you can do with WMI comes down to memorizing every available class—except that there are *hundreds* of classes, and the number grows with each new Microsoft product release. The best place to start to research what you can do with WMI is the MSDN Library at <http://msdn.microsoft.com/library>. Browse to Setup and System Administration, Windows Management Instrumentation, SDK Documentation, Windows Management Instrumentation, WMI Reference, WMI Classes. This path changes often, so you might need to perform a search.

You'll find sections on Win32 classes, which are useful, and registry classes, which are also useful. Figure 3.6, for example, shows the `Win32_OperatingSystem` class documentation. As you can see, it lists all of the class' properties as well as its four methods: `Reboot`, `SetDateTime`, `Shutdown`, and `Win32Shutdown`.

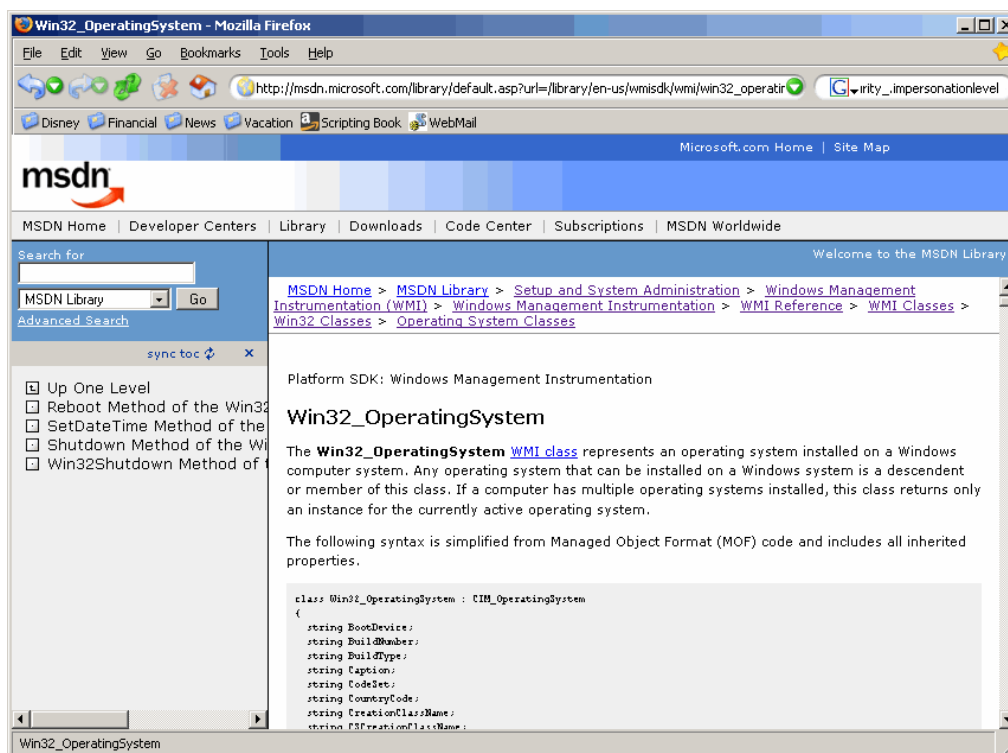


Figure 3.6: Exploring the WMI documentation.

Given the documentation, you can start to see how all WMI scripts start to look alike. For example, consider the script that Listing 3.4 shows, which writes out the current service pack version of each computer in the domain (this script uses the template script used in Chapter 2).

```

`connect to the root of AD
Dim rootDSE, domainObject
Set rootDSE=GetObject("LDAP://RootDSE")
domainContainer = rootDSE.Get("defaultNamingContext")
Set oDomain = GetObject("LDAP://" & domainContainer)

Dim oFSO, oTS
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oTS = oFSO.CreateTextFile("servicepacks.txt", True)

`start with the domain root
WorkWithObject(oDomain)

oTS.Close
MsgBox "Done!"

Sub WorkWithObject(oContainer)
  Dim oADObject
  For Each oADObject in oContainer
    Select Case oADObject.Class
      Case "user"
        `oADObject represents a USER object;
        `do something with it
        `** YOUR CODE HERE**
      Case "computer"
        strComputer = ADSObject.cn
        `oADObject represents a COMPUTER object;
        `do something with it
        `connect to the computer via WMI
        Set oWMIService = GetObject("winmgmts:" _
          & "{impersonationLevel=impersonate}!\\" & _
          strComputer & "\root\cimv2")

        `retrieve OS class
        Set oWindows = oWMIService.ExecQuery("SELECT * " & _
          "FROM Win32_OperatingSystem")

        `output sp level
        For Each oOS in oWindows
          oTS.WriteLine oADObject.Name & ": " & _
            oOS.Name & " / " & oOS.Version & " SP" & _
            oOS.ServicePackMajorVersion & "." & _
            oOS.ServicePackMinorVersion
        Next

        Case "organizationalUnit" , "container"
          `oADObject is an OU or container...
          `go through its objects
          WorkWithObject(oADObject)
        End select
      Next
    End Sub

```

Listing 3.4: Example script that writes out the current service pack version of each computer in the domain.

Suppose that you wanted to see how much free space was available on every computer's disk drives. A very similar script emerges; Listing 3.5 shows the new script with the few lines that have been changed from the previous example script in boldface.

```

`connect to the root of AD
Dim rootDSE, domainObject
Set rootDSE=GetObject("LDAP://RootDSE")
domainContainer = rootDSE.Get("defaultNamingContext")
Set oDomain = GetObject("LDAP://" & domainContainer)

Dim oFSO, oTS
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oTS = oFSO.CreateTextFile("diskspace.txt", True)

`start with the domain root
WorkWithObject(oDomain)

oTS.Close
MsgBox "Done!"

Sub WorkWithObject(oContainer)
  Dim oADObject
  For Each oADObject in oContainer
    Select Case oADObject.Class
      Case "user"
        `oADObject represents a USER object;
        `do something with it
        `** YOUR CODE HERE**
      Case "computer"
        strComputer = ADSObject.cn
        `oADObject represents a COMPUTER object;
        `do something with it
        `connect to the computer via WMI
        Set oWMIService = GetObject("winmgmts:" & _
          & "{impersonationLevel=impersonate}!\\" & _
          strComputer & "\root\cimv2")

        `retrieve OS class
        Set oDisks = oWMIService.ExecQuery("SELECT * " & _
          "FROM Win32_LogicalDisk")

        `output sp level
        For Each oDisk in oDisks
          oTS.WriteLine oADObject.Name & ": " & _
            oDisk.Name & " has " & oDisk.FreeSpace & " bytes free"
        Next

      Case "organizationalUnit" , "container"
        `oADObject is an OU or container...
        `go through its objects
        WorkWithObject(oADObject)
    End select
  Next
End Sub

```

Listing 3.5: Example script to see how much free space was available on every computer's disk drives.

Once you become accustomed to working with WMI, it all starts to look a lot alike. This repetition is great because it means that the time you take to write one WMI-based script will save you a lot of time the next time you need to write one.

WMI Scriptlets

At this point, you're probably itching to see some more WMI in action. WMI can be useful for querying information, changing settings, and performing actions such as restarting a computer. The next few sections will provide scriptlets that demonstrate the possibilities. Keep in mind that these may require some minor modifications—such as different server names—in order to run properly in your environment; don't try to cut and paste them and expect them to work right away in every environment.

Managing Services

Services are something we would all just as soon forget about. They're not readily managed from a central location and they are difficult to manage on a server-by-server basis. VBScript can help, as Listing 3.6 shows.

```
'connect to the computer's WMI provider
sComputer = "server1"
Set oWMIService = GetObject("winmgmts:" & _
    & "{impersonationLevel=impersonate}!\\" & _
    sComputer & "\root\cimv2")

'retrieve the MyApp service
Set oService = oWMIService.ExecQuery _
("Select * from Win32_Service WHERE Name" & _
    " = 'myservice'")

'change the password
errReturn = oService.Change( , , , , , , _
    , "NeWP@ss#0rD")
```

Listing 3.6: An example script that helps in managing services.

This script connects to WMI on a server named Server1, using WMI impersonation to impersonate the credentials of whoever is running the script. It executes a WMI query that is a bit unlike any you've seen so far. Rather than querying *every* instance of Win32_Service, it queries all instances whose Name property is myservice. You could make that Alerter, Messenger, or some actual service name. The script then executes the service's Change() method, skipping the first seven parameters, which are things we don't want to change right now, and passes in a new password. This new password becomes the password that the service will use to log on. This scriptlet is a great way to update a service to use a new password, making it easier to regularly change service account passwords.

👉 If you get an error indicating that "Object doesn't support this property or method: 'oService.Change'," the odds are that oService isn't set to an instance of the Win32_Service class. That can—and will—happen if your query doesn't specify a valid service name.

Of course, this script only works against one computer; it would be more useful if it could run against a list of computers running this particular service. Say, a text file with one computer name per line. Listing 3.7 shows an example script that does so.

```

'open a text file of computer names
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oTS = oFSO.OpenTextFile("c:\computers.txt")

Do Until oTS.AtEndOfStream

  'get computer name
  sComputer = oTS.ReadLine

  'connect to the computer's WMI provider
  Set oWMIService = GetObject("winmgmts:" _
    & "{impersonationLevel=impersonate}!\\" & _
    sComputer & "\root\cimv2")

  'retrieve the MyApp service
  Set oService = oWMIService.ExecQuery _
    ("Select * from Win32_Service WHERE Name" & _
    " = 'myservice'")

  'change the password
  errReturn = oService.Change( , , , , , _
    , "NeWP@ss#0rD")
Loop

oTS.Close

```

Listing 3.7: An example script that manages a list of computers running a particular service.

In Listing 3.7, the modified lines are highlighted in boldface so that you can see what a relatively minor change this is. Want the script to provide feedback about what's going on? You need to make only one simple change, as Listing 3.8 shows.

```

'open a text file of computer names
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oTS = oFSO.OpenTextFile("c:\computers.txt")

Do Until oTS.AtEndOfStream

  'get computer name
  sComputer = oTS.ReadLine

  'connect to the computer's WMI provider
  Set oWMIService = GetObject("winmgmts:" _
    & "{impersonationLevel=impersonate}!\\" & _
    sComputer & "\root\cimv2")

  'retrieve the MyApp service
  Set oService = oWMIService.ExecQuery _
    ("Select * from Win32_Service WHERE Name" & _
    " = 'myservice'")

  'change the password

```

```

errReturn = oService.Change( , , , , , _
, "NeWP@ss#0rD")

`message
WScript.Echo "Changed " & sComputer

Loop

oTS.Close

```

Listing 3.8: The same example script with a modification added to enable feedback.

What Happened to Dim?

The first chapter stated that the Dim statement was required to tell VBScript that you were planning on using a particular variable. Well, “required” is a strong term. As you can see from the past few samples, scripts run fine without it. The truth is that Dim is optional—you don’t have to tell VBScript up front. If VBScript encounters a variable that it hasn’t seen before, it implicitly declares it for you, saving you time. However, Dim is a useful tool that can prevent errors from being overlooked. Read the following script sample carefully:

```

sComputer = InputBox("Computer name?")
sPassword = InputBox("New password for service?")
Set oWMIService = GetObject("winmgmts:" _
& "{impersonationLevel=impersonate}!\\" & _
sComputer & "\root\cimv2")
Set oService = oWMIService.ExecQuery _
("Select * from Win32_Service WHERE Name" & _
" = 'myservice'")

`change the password
errReturn = oService.Change( , , , , , _
, sPasswpd)

```

Spot the problem? The password you entered is stored in the variable sPassword, but you made a typo in the last line and actually used the contents of variable sPasswpd to set the new password. VBScript hasn’t seen sPasswpd before, so it initializes a new, empty variable, and now your service is screwed up. To avoid such mistakes, use Dim:

```

Option Explicit
Dim sComputer, sPassword, oWMIService, oService
sComputer = InputBox("Computer name?")
sPassword = InputBox("New password for service?")
Set oWMIService = GetObject("winmgmts:" _
& "{impersonationLevel=impersonate}!\\" & _
sComputer & "\root\cimv2")
Set oService = oWMIService.ExecQuery _
("Select * from Win32_Service WHERE Name" & _
" = 'myservice'")

`change the password
errReturn = oService.Change( , , , , , _
, sPasswpd)

```

This modified script will result in an error on the last line of code because you didn’t declare sPasswpd. The Option Explicit statement at the beginning of the script tells VBScript to disallow implicit variable creation and to require variables to be announced via the Dim statement. Now you’ll spot your typo immediately, thanks to the error, without screwing up any of your service settings.

Now that you have this service-changing example in hand, the next thing you should do is scurry into the WMI documentation to find out what *else* this script might be able to do. I've already mentioned that the Win32_Service class' Change() method has at least seven other arguments: What do they do?

According to the documentation, I can use this method to change:

- The display name
- The path name
- The service type
- The error control type
- The start mode
- Whether desktop interaction is allowed
- The login account
- The login password
- The order in which the service should load
- The dependencies the service has

Whenever you get a WMI example, spend some time exploring what else you can do with the objects, classes, or whatever else you've been introduced to. It's a great way not only to accomplish a current task but also to incrementally learn more and more about VBScript and WMI.

Archive Security Logs

Still waiting for the Microsoft Audit Collection Server (MACS) to show up on your network? Listing 3.9 provides a handy script that will read a text file of computer names, connect to each one, archive its security log to a local file (on the computer running the script, that is), then clear the log to make room for new entries.

```

`Create a FileSystemObject
Set oFS = CreateObject("Scripting.FileSystemObject")

`Open a text file of computer names
`with one computer name per line
Set oTS = oFS.OpenTextFile("c:\computers.txt")

`go through the text file
Do Until oTS.AtEndOfStream

    `get next computer
    sComputer = oTS.ReadLine

    `connect to the WMI provider
    Set oWMIService = GetObject("winmgmts:" _
    & "{impersonationLevel=impersonate,(Backup,Security)}!\\" & _
    sComputer & "\root\cimv2")

    `query the Security logs
    Set cLogFiles = oWMIService.ExecQuery _
    ("Select * from Win32_NTEventLogFile where " & _
    "LogFileName='Security'")

    `go through the collection of logs
    For Each oLogfile in cLogFiles

        `back up the log to a file
        errBackupLog = oLogfile.BackupEventLog _
        ("c:\logfile\" & sComputer & "\" & Date() & ".evt")

        `see if an error occurred
        If errBackupLog <> 0 Then

            `one did - display an error
            Wscript.Echo "Couldn't get log from " & sComputer

        Else

            `no error - safe to clear the Log
            oLogfile.ClearEventLog()


        End If
    Next
Loop

`close the input file
oTS.Close

```

Listing 3.9: An example script that reads a text file of computer names, connects to each one, archives its security log, then clears the log to make room for new entries.

Notice the impersonation permission being requested: Backup,Security. This permission is necessary to archive security logs. The script selects every instance of Win32_NTEventLogFile where the name of the log file (the LogFileName property) is Security. It's theoretically possible for that query to return multiple instances of Win32_NTEventLogFile, so you must assume the result of the query will be a collection. Therefore, it makes sense to use a For Each...Next construct to loop through each instance one at a time.

 If you are receiving access denied errors, your system may be configured to require other special permissions. Generally, the Backup permission suffices for log files, although the Security log in particular often requires the additional Security permission. Your system may be configured to require additional permissions.

Each instance's BackupEventLog() method is called. The only argument this method requires is a filename, so I built one that includes the remote computer's name, along with the current date, for easier identification later. The method returns a value greater than zero if the backup didn't work; specifically, it returns the value 8 if permission is denied, 21 if the archive path is invalid, and 183 if the archive path already exists.

If the value returned is zero, the script clears the log using the ClearEventLog() method; if not, the script outputs an error message indicating that the log couldn't be backed up. Note that in some instances, you may need to pass the path and filename of the event log as an argument of ClearEventLog(); if this is necessary, you can modify the script as follows:

```
'no error - safe to clear the Log
oLogFile.ClearEventLog(oLogFile.Path & oLogFile.FileName)
```

Extended WMI

Another great way to fiddle with WMI and see what it can do is by using the Tweakomatic, another tool from the Microsoft Scripting Guys. You can access this tool at <http://www.microsoft.com/technet/community/scriptcenter/tools/twkmatic.msp>. For example, if I wanted to modify a computer's screen saver to require a password, the Tweakomatic tells me that I could use the short script that Listing 3.10 shows.

```
HKEY_CURRENT_USER = &H80000001
strComputer = "."
Set objReg = GetObject("winmgmts:\\\" & _
  strComputer & "\\root\default:StdRegProv")
strKeyPath = "Control Panel\Desktop"
objReg.CreateKey HKEY_CURRENT_USER, strKeyPath
ValueName = "ScreenSaverIsSecure"
strValue = "1"
objReg.SetStringValue HKEY_CURRENT_USER, _
  strKeyPath, ValueName, strValue
```

Listing 3.10: An example Tweakomatic script to modify a computer's screen saver to require a password.

Notice anything different about the WMI query that Listing 3.10 shows? The difference is in the WMI query, which uses root\default:StdRegProv rather than root\cimv2. What's going on there?

This is where WMI and quantum physics really start to look interchangeable. Microsoft only crams certain WMI classes into the root\cimv2 namespace. Other WMI classes get crammed elsewhere. In this example, it's the Standard Registry Provider that the script is working with, so you must connect to its namespace. Exchange Server 2000 and Exchange Server 2003 install WMI classes, and they use the root\cimv2\applications\exchange namespace. Or they did. In Exchange Server 2000 SP2 and later, Microsoft started the root\cimv2\applications\exchangev2 namespace, which includes providers named ExchangeDsAccessProvider and ExchangeMessageTrackingProvider.

IIS gets in on the game, too, with the MicrosoftIISv2 namespace (root\microsoftiisv2). SQL Server 2000 has the root\MicrosoftSQLServer namespace. Other Microsoft products add WMI classes, too, and it's cursedly difficult to find them because Microsoft doesn't document them in one convenient place. BizTalk Server uses root\MicrosoftBizTalkServer, for example, and I don't know where you would dig up that information without a search. There is a WMI namespace for SNMP providers, allowing you to interact with the providers via WMI. You can install and uninstall applications using the Windows Installer namespace, and there is even a namespace for performance counters, so you can write scripts that retrieve performance information. Unfortunately, WMI is so broad in scope that it would be impossible to address them all in this guide.

You can use the WBEMTEST tool, or even the WMI Scriptomatic, to browse the available classes (to a point; they don't query every available namespace). But those tools will only list classes available on the *local machine*; thus, if you want to see things like Exchange classes, you have to run the tool on a machine that contains those classes—namely, an Exchange Server system.

If you're eager to see what namespaces are available on a particular computer, run the following script (I recommend running it under CScript because it outputs a lot of information):

```
sComputer = "."
Call EnumerateNamespaces("root")
Sub EnumerateNamespaces(sNamespace)
    WScript.Echo sNamespace
    Set oWMIService = GetObject("winmgmts:\\\" & _
        sComputer & "\" & sNamespace)
    Set cNamespaces = oWMIService.InstancesOf("__NAMESPACE")
    For Each oNamespace In cNamespaces
        Call EnumerateNamespaces(sNamespace & _
            "\" & oNamespace.Name)
    Next
End Sub
```

My WS2K3 machine returns the following interesting entries, giving you some idea of the scope WMI encompasses:

- Root\SECURITY—for working with security
- Root\perfmon—for working with performance counters
- Root\RSOP—for working with Resultant Set of Policy (RSOP) in Group Policy
- Root\snmp—for working with SNMP
- Root\MSCluster—for working with Microsoft Cluster Server
- Root\cimv2—for working with core Win32 classes
- Root\Applications\MicrosoftIE—for working with Internet Explorer
- Root\MicrosoftActiveDirectory—for working with Active Directory
- Root\MicrosoftIISv2—for working with IIS 6.0
- Root\Policy—for working with policies
- Root\MicrosoftDNS—for working with DNS
- Root\MicrosoftNLB—for working with Network Load Balancing
- Root\registry—for working with the registry

If you repeat this chapter's very first exercise, using WBEMTEST, and substitute these namespace for root\cimv2, you'll be able to check out the classes in these namespaces. Instead of clicking Query, click Enum Classes. Leave the Superclass name blank, click Recursive, and click OK. You should get a list of all available classes in the namespace.

Summary

WMI, ADSI, and VBScript work together to provide a well-rounded, robust set of administrative tools. Between them, there is not much you can't do with regard to remote management. However, there are some advanced techniques that can make scripting a bit more powerful, and I'll take a look at some of them in the next chapter.