

Are you WMI or Not? (Part I)

by Craig Murphy

Developers thrive on avoiding mundane repetitive tasks – they thrive on writing some code to help perform the tasks. Over the course of two articles, I will be demonstrating how Microsoft® Windows® Management Instrumentation (WMI) has helped me automate the creation of scheduled tasks and how it has made my life a little easier.

Introduction

As most managers will know, developers will go out of their way to avoid repetitive tasks. So much so that they will often spend hours researching a subject so that they can write some code to avoid the mundane chore of a repetitive task. The repetitive task may only take a couple of hours to complete, but that doesn't stop a developer trying to automate it! I recently found myself in this very position – I had to create a number of scheduled tasks. Creating the scheduled tasks isn't that repetitive, but I had to replicate those tasks on a number of machines, which is when I turned to the Microsoft® Windows® Management Instrumentation (WMI).

Motivation

Over the last few months, amongst lots of other things, I have been involved in the rollout of our corporate anti-virus solution. With the availability of virus creation toolkits and an ever-increasing base of malicious virus developers, keeping personal and corporate anti-virus solutions up to date is of paramount importance.

The anti-virus solution that I have been using allows server-side and client-side updating via the vendor's web site. As new viruses are discovered, new signature files are made available. The new signature files are available as a download from the vendor's web site and can be sent via e-mail. Indeed, even minor updates to the product itself are distributed in this manner.

However, both approaches require human intervention – the new signatures have to be downloaded and distributed. In today's world where we seem to be getting busier and busier, relying on a human to download and distribute signature files is fraught with danger and does not make economic sense.

So, to download new virus signatures and product updates reliably and in a timely fashion, I use a routine that downloads particular files from the vendor's web site. The routine in question is nothing more than a DOS batch file that is executed via a scheduled task. Once this scheduled task has executed successfully, a further nineteen scheduled tasks distribute parts of the downloaded files to various remote locations within our corporate WAN. I use a separate scheduled task for each location in order to be able to track any failures. If I had one batch file that copied the files to each location, how would I be able to determine which location(s) were updated or not?

As we will see later in this article, setting up one scheduled task is not a chore – however setting up twenty scheduled tasks is! Then to have to set up the same twenty tasks on another computer just makes it even more of a chore.

What is WMI?

WMI is a mechanism that allows programmatic access to virtually all of the Windows resources you could possibly want to work with. The Windows resources that I'm talking about include such things as system information, BIOS information, boot configuration, installed codecs, drive partition information, environment variables, operating system information – the list goes on. Microsoft has produced a chart of the WMI classes that are provided – it fits onto a sheet of A3...and covers less than 20% of the WMI classes that are available. If you would like to see this chart, search the Microsoft Developer Network (MSDN) web site for 'wmichart'.

If you examine the task list on your computer, you should find a process called WinMgmt.exe – it is this process that is the core of WMI. WinMgmt.exe is known as the Computer Information Model Object Manager (CIMOM).

The CIMOM maintains a database or repository that contains object definitions, class and instance definitions that can be used to access and maintain system configuration information. The CIMOM database has to be populated – this is achieved by Managed Resources, such as Windows itself, a Windows service, or even an application, providing CIMOM with information about the API that it would like WMI to be aware of. In the first sentence of this paragraph I used the word 'database'. Most databases have query languages built in and, true enough, the query language that WMI uses is WQL – Windows Management Instrumentation (WMI) Query Language. I will discuss WQL in part two of this article.

Sitting in between the Managed Resource and the CIMOM is the WMI Provider. The WMI Provider is essentially a marshalling layer that provides a consistent interface between the Managed Resource and the CIMOM. This article will be concentrating on using WMI (consuming), rather than building WMI-compliant applications – hence we will not go in to much detail as to how this interface actually works.

The final part of the equation is something that uses the WMI classes – a WMI consumer. A WMI consumer can take many forms – it could be an ASP page, a Visual Basic Script (VBS), or a Delphi application. If you have ever run a software tool that audits your PC, it is very likely that it used WMI classes to access system information – in which case the audit software is considered a WMI consumer. Equally, the "System Info..." button found in Help/About Microsoft Word (for example) is a WMI consumer. Figure 1 presents the Elementary WMI Architecture.



Figure 1 – Elementary WMI Architecture

WMI – A Simple Example

I will apologise now – this is going to be a Visual Basic Script (VBScript) example. WMI classes can be consumed using scripting languages (via the Scripting API for WMI) and via the COM API for WMI. Using the COM API for WMI requires us to write a reasonable amount of code – and that is something that we will do in the second article. However, with so many of us becoming multi-lingual, and the relative simplicity of VBScript, the examples should not present you with any problems.

As most developers will agree, when we ask an end user the question “What OS are you using?” the replies we receive range from precise to generic...from Windows 2000 SP2 down to just Windows. The generic response is not much good if you are trying to work out why your application refuses to work on their machine! So, as Microsoft has demonstrated, including some system information in your application’s Help/About dialog can be a useful time saver.

There are many ways of obtaining this information, the most obvious being to trawl the Windows Registry. This would work, however the format of the registry settings varies from Windows 95 to Windows XP. Tapping into the registry for information is not always that easy - network card information is stored via the use of linked GUIDs. Incorporating GUID tracing code into your application just adds to the complexity. To avoid these problems, we can make use of Microsoft’s recommended WMI to use a standard, consistent interface that allows us to manage the entire machine’s information/resources regardless of which OS is installed.

We have already seen that WMI is based around the concept of providers. A core provider is known as the “Win32 provider”, and is implemented by cimwin32.dll. The Win32 provider is able to supply information about the computer, disks, peripheral devices, files, folders, file systems, networking components, operating system, printers, processes, security, services, shares, and much more.

There are literally hundreds of WMI classes available – these are organised into groups that are identified using namespaces. The Win32 provider’s classes are found in the “root\cimv2” namespace – which includes the classes that provide information about the computer and installed OS. As you might expect, there is the concept of a default namespace. The default namespace is typically root\cimv2, which is why it is omitted from the listings in this article.

However, for the sake of completeness, listing 1 demonstrates how to use WMI with or without a namespace.

Listing 1 presents the Visual Basic Script required to extract a variety of OS-specific facets. Most of the facets (ServicePackMajorVersion, OSType, etc.) are self-explanatory.

```

' Listing1.vbs
' Listing1.vbs
' demonstrates extractions of OS info
' using WMI

strComputer = "."

' use the default namespace
Set objWMIService = _
  GetObject("winmgmts:\\\" & strComputer)

' use the specified namespace
' Set objWMIService = GetObject( _
' "winmgmts:\\\" & strComputer & "\root\cimv2")

Set colOperatingSystems = _
  objWMIService.InstancesOf( _
  "Win32_OperatingSystem")

For Each objOpSys In colOperatingSystems
  Wscript.Echo _
    "Name: " & objOpSys.Name & vbCrLf & _
    "Caption: " & objOpSys.Caption & vbCrLf & _
    "CurrentTimeZone: " & objOpSys.CurrentTimeZone & vbCrLf & _
    "LastBootUpTime: " & objOpSys.LastBootUpTime & vbCrLf & _
    "LocalDateTime: " & objOpSys.LocalDateTime & vbCrLf & _
    "Locale: " & objOpSys.Locale & vbCrLf & _
    "Manufacturer: " & objOpSys.Manufacturer & vbCrLf & _
    "OSType: " & objOpSys.OSType & vbCrLf & _
    "Version: " & objOpSys.Version & vbCrLf & _
    "Service Pack: " & objOpSys.ServicePackMajorVersion & _
    "." & objOpSys.ServicePackMinorVersion & _
    & vbCrLf & _
    "Windows Directory: " & objOpSys.WindowsDirectory
Next
  
```

Listing 1 – Extracting OS info using WMI

Assuming that you execute Listing1.vbs by double clicking on it, figure 2 presents the output. Alternatively, if you wish to execute Listing1.vbs via the command-line, simply entering “cscript Listing1.vbs” at the command-line will produce similar results.

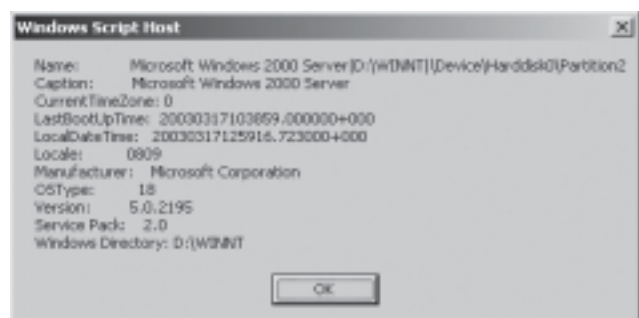


Figure 2 – Output from Listing 1

Not rocket science, I know, but useful nonetheless. Listing 1 presents a couple of interesting concepts. Firstly, what is “winmgmts:”? This is a moniker, an alias, for the WMI Scripting Library – it is this library that gives us access to a computer’s resources. In addition to the WMI Scripting Library, we must specify the name of the computer we wish to connect to – in this case we used ‘.’ which represents ‘this’ computer or the local connection. In part two of this article I will demonstrate how we can use WMI to connect and retrieve information from remote computers on the same network.

Once we have access to the WMI Scripting Library, we can create InstancesOf particular Win32_ classes – in this case Win32_OperatingSystem. The WMI Scripting Library returns a collection – you can probably start to see why this might require a little bit of work in a strongly typed environment such as Delphi. Iterating over these items in this collection, of which in my case there is only one, allows us to extract the OS information.

Creating Scheduled Tasks

Now that we have covered the basics of WMI, let us take a look at how I solved my original problem – the need to create Scheduled Tasks programmatically. The manual Scheduled Tasks functionality can be found in the Control Panel. In my opinion, scheduled tasks are something you either love or hate. I used to hate them, but have recently grown to like them.

Listing 2 presents the code that I used to create a scheduled task that executes every Monday and Friday at 1335. The task that is executed is BATCH.BAT. There is no reason why the task that is executed could not be a Delphi application – my original problem was solved by simply copying files from one location to another, hence the use of a DOS batch file.

```

\ Listing2.vbs
\ Creating a scheduled task

\ Define week-day contants
bMonday = 1
bTuesday = 2
bWednesday = 4
bThursday = 8
bFriday = 16
bSaturday = 32
bSunday = 64

strComputer = "."
fError = false

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

Set objNewJob =
    objWMIService.Get("Win32_ScheduledJob")

call add_task( "C:\BATCH.BAT", _
    "*****133500.000000-000", _
    bMonday OR bFriday)

if fError = false then WScript.Echo "Scheduled
    tasks created!"
\

```

```

Public Function add_task(strTask, strTime, bDOW)

    errJobCreated = objNewJob.Create(strTask,
        strTime, _
        True , bDOW, , _
        True, JobID)

    If errJobCreated <> 0 Then
        Wscript.Echo "Error during task creation"
        fError = true
    End If

End Function

```

Listing 2 – Creating Scheduled Tasks

Whilst reading listing 2, keep in mind that the ‘_’ character indicates to the VBScript interpreter that a line of code is continued on the next line.

To make my life a little easier, and to make listing 2 a little more readable, I created a function add_task. This function takes three parameters: the task to execute (a batch file in this case), a time at which execute and an integer representing the day(s) on which to execute. The first parameter is easy to explain, it is simply the name of the batch file. The third parameter uses a Boolean build-up to create a number that represents the day or days of the week the task is to be executed. I have defined some constants at the top of listing 2 that provide the day to day integer mapping, ORing them together allows us to specify multiple days of the week.

The second parameter is a little more complicated. However, the complexity is soon reduced after consulting the help that is available for the Win32_ScheduledJob provider. The help states:

“The StartTime property represents the UTC time to run the job, in the form of YYYYMMDDHHMMSS.MMMMMM(+/-)OOO, where YYYYMMDD must be replaced by *****. The replacement is necessary because the scheduling service only allows jobs to be configured on a day of the month, day of the week, or run once. A job cannot be run on a specific date.

Example: *****123000.000000-420 which implies 12:30 pm PST with daylight saving time in effect.”

There are copious quantities of help available for most of the WMI providers. Later in this article, in the section WMI Administration Tools, I will show you how we can extract the help information.

As you can see, the add_task function lets me create any number of scheduled tasks with a simple double-click of the mouse. Better still, when I need to replicate those scheduled tasks on another machine I have two options. Firstly, I could simply copy Listing2.vbs onto the other machine then double-click on it. Or secondly, I could change the contents of strComputer to point to the other machine (assuming both were on the same network). Thus I could achieve my end result without ever leaving my desktop (even to the point of not using pcAnywhere to execute the script on a remote computer!) I will cover the latter approach in part two of this article.

Running Listing2.vbs adds a new scheduled task that is run using the old DOS “at” command. Figure 3 presents a screenshot of the Scheduled Task list after Listing2.vbs has been double-clicked on.



Figure 3 – Successfully adding a Scheduled Task

There is one caveat however – the “at” command creates scheduled tasks that need to be executed using a particular user’s permissions. This can be one of the main reasons an “at” command scheduled task is created successfully, but then fails to execute. To avoid this issue, the Scheduled Task list control offers a means of specifying the “at” command’s service account. As can be seen in figure 4, the ‘Advanced’ menu option has an ‘AT Service Account’ option.

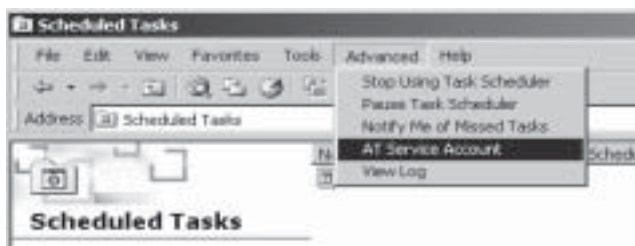


Figure 4 – Modifying the AT Service Account

Specifying the AT Service Account is a fairly painless process. By default it will use the System Account. However, in some cases it may be necessary to use another account – perhaps with different rights and privileges. Figure 5 presents the AT Service Account configuration.



Figure 5 – Changing the AT Service Account

Is WMI a Windows 2000 or XP thing?

Thankfully not – whilst WMI is the “core management-enabling technology” in Windows 2000, Windows XP and Windows Server 2003, it is also available for Windows ME, Windows 98 and Windows 95 (OSR 2.5). Out of the box, Windows 2000, XP and Windows Server 2003 have WMI 1.5 pre-installed. If you need to install WMI on the older operating systems, there is a free download available via the MSDN web site – there is a link available in the Resources section of this article.

WMI Administrative Tools

If this article has captured your attention and you would like to explore WMI in more detail, the WMI Administrative Tools will prove invaluable. In the Resources section of this article I have provided the url of the relevant MSDN web site – all the WMI downloads are available from here, including the Windows ME, 98 and 95 WMI installations.

The WMI Admin Tools is a 4.5mb download from the MSDN web site, so even in a “broadband denied” location it is not a painful download. If you have downloaded and installed the WMI Admin Tools, you will have noticed a new program group has been created in the Programs group – WMI Tools. The WMI Admin Tools consist of four tools, as shown in figure 6. The primary tool that we can use to help us understand WMI is the WMI CIM Studio. CIM Studio is a graphical front-end that allows us to navigate the WMI classes on both local and remote computers.



Figure 6 – The WMI Administration Tools

CIM Studio can provide us with a wealth of information. Using familiar controls, such as the traditional Microsoft binoculars representing Search, we can quickly locate the Win32_ScheduledJob WMI class. Figure 7 presents a screenshot of CIM Studio analysing Win32_ScheduledJob.

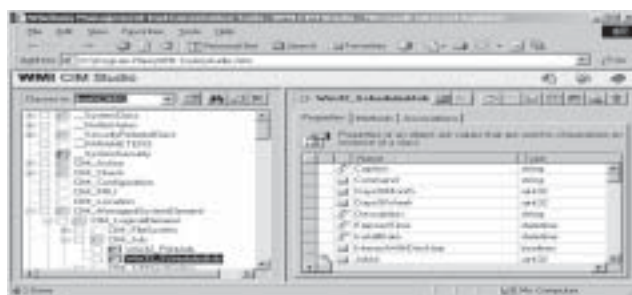


Figure 7 – CIM Studio analyses Win32_ScheduledJob

CIM Studio is also a great source of WMI help. On the right hand side of the CIM Studio screenshot, there is a question mark icon – clicking on this invokes pop-up help. If we think back to listing 1 where we obtained information about the OS, figure 8 presents the help that we would expect to see for the Win32_OperatingSystem provider.



Figure 8 – Win32_OperatingSystem help

And finally...

That is all for part one. Hopefully I have shown you enough to start thinking about WMI as a means of incorporating machine management functionality into your applications. In part two, I will show you how we can use WMI in a Delphi application. I will also demonstrate how WMI can be used over a network i.e. being able to remotely interrogate another computer.

If you liked some of what you have read here but do not wish to download the WMI Admin Tools, then try running the WMI Tester – you can do this by typing “wbemtest” in a command prompt window. WBEM stands for Web-Based Enterprise Management – WMI is Microsoft’s implementation of WBEM. WBEM itself is supposed to be a vendor-neutral and protocol-independent standard for managing computer infrastructure. The WMI Tester is a general-purpose, graphical tool for interacting with the WMI infrastructure: it is not as intuitive as the WMI Admin Tools, but it will give you a good idea of just how vast the WMI infrastructure actually is.

What’s Next?

In the next part of this article I will cover the following topics:

- Using WMI to retrieving information from a remote computer
- Using WMI in Delphi
- The WMI Query Language (WQL)

Resources

MSDN WMI Downloads: <http://msdn.microsoft.com/library/default.asp?url=/downloads/list/wmi.asp>



Craig is an author, developer, speaker and Dilbert Evangelist – he specialises in all things XML, particularly SOAP and XSLT. He can be reached via e-mail at: Craig@isleofjura.demon.co.uk, or via his web site: <http://www.craigmurphy.com> (where you can also find the source code and PowerPoint files for all of Craig’s UK-BUG articles and presentations).

Tips from the support team

If you’re not sure what format the string representation of the date is in, you can try:

```
var
  strDate: string;
  dDate: TDateTime;
begin
  ...
  dDate := VarAsType(strDate, varDate);
  ...
end;
```

Be careful to check which way it interprets “03-04-05”, of course (my machine will interpret that as 3rd April 2005 but I suspect that’ll not be the same for anyone); but it will happily have a go at converting a whole lot of formats, including “2-Mar-03”, “2 March 2003”, “March 2 2003”, etc - and seems happy with either “-”, “,” or “/” as a date separator (though periods are rejected).