WMI Without Scripting (Well, Almost...)
July 16, 2002

by Marcin Policht

Even though interest in automating administrative tasks via scripting grows steadily, it will still be a while before Windows Scripting Host, Active Directory Services Interface, or Windows Management Instrumentation become common items in administrative toolkits on par with standard Administrative Tools included in Windows 2000 or XP. Learning intricate object models used by ADSI or WMI is without doubt a challenging task, especia those who lack prior programming or scripting experience. One of Microsoft's recent attempts in addressing this challenge is WMIC. This acronym stands for Windows Management Instrumentatic Command Line Tool, which is a component available in Windows XP and the soon-to-be released Windows .NET server platform. WMIC offers a more friendly, non-scripted way of using Windows Management Instrumentation for managing computing environment.

Even without the need for scripting, WMIC is a bit of a challenge, since the documentation for it is, most part, limited and sometimes even inaccurate. In this article, I'll describe ways of using WMIC process management. As you will find out, WMIC was designed to give you most of the power and flexibility associated with WMI.

Let's start with the basics. WMIC works in two modes - interactive and non-interactive. Interactive initiated by running WMIC at the Command Prompt or from the Start->Run box. In either case, you redirected to the **wmic:root\cli>** prompt ("root\cli" is the name of the WMI namespace where WMI functionality has been implemented). Non-interactive mode involves prefixing each of the comman the word WMIC, and can be run directly from the command prompt (which makes it a perfect cand for batch files).

From either mode, you can type /**?** to find out the list of available global switches and aliases (whic commands you use to interact with Windows Management Interface objects).

Global switches

Global Switches are used to set the properties of the WMIC environment. For example, /**NAMESP** allows you to change the namespace you work with. (Namespaces are groupings of WMI classes, in turn represent managed software and hardware objects, such as processes, services, installed applications, processors, memory, etc. Most of the time, you can use the default root/cimv2 namespace.) /**USER** and /**PASSWORD** allow you to set alternate credentials (other than the curre logged on user) for connecting to WMI. The /**NODE** switch is used to specify the computer (or com that will be the target of the WMI commands. To change the node you want to access, you would t from the command prompt

```
WMIC /NODE:"USLT-NYPZ0005"
```

(remember to enclose the switch value in double quotes if the value contains special characters lik '/').

The argument of the /**NODE** switch can be also a file containing the computer names (one name p e.g.

```
/NODE:@"C:\Swynk\WMIC\PCLIST.TXT"
```

If you want to record the output of the commands you will be executing, you can use the /**RECORI** switch. This is done by typing the following:

```
/RECORD:"C:\Training\Swynk\WMIC\OUTPUT.TXT"
```

This option is especially useful since WMIC tends to generate a large amount of output. For examp **PROCESS LIST** command, in its **FULL** format, lists the values of forty-one process properties (suc CommandLine, Description, ExecutablePath, ExecutionState, Handle, HandleCount, InstallDate, KernelModeTime, MaximumWorkingSetSize, MinimumWorkingSetSize, Name, PageFaults, PageFileUsage, ParentProcessId, PeakPageFileUsage, PeakVirtualSize, PeakWorkingSetSize, Pı and ProcessId, just to mention the first few).

You can find out the current values of all global switches by typing **CONTEXT** from the **wmic:root** prompt) or **WMIC CONTEXT** from the Command Prompt.

Aliases

Aliases represent managed objects and provide a more friendly way of accessing WMI classes. Fc example, Win32_Memory class is referenced by the **MEMORY** alias, Win32_Product class corresp to the **PRODUCT** alias (which is used to manage Windows Installer packages), and the Win32_Pr class can be accessed using the **PROCESS** alias. As you can see, names of classes can be easil matched with their equivalent aliases.

Since reviewing all available aliases is not feasible in one short article, I will focus on using WMIC process management.

Listing processes

PROCESS LIST - allows you to enumerate processes on the target node

This command can be used with several parameters that will modify the format and content of the The relevant ones (from a system administration point of view) would be **BRIEF** (limiting the displa few relevant properties such as Name, Priority, ProcessID, and WorkingSetSize), **IO** (listing proper relating to Input/Output operations), **MEMORY** (presenting the impact of each process on memory statistics) and **STATUS** (the most succinct one providing only Name, ProcessID, and Status).

In addition, you can use the **WHERE** statement to further customize the display. For example, to d
all instances of the Command Prompt on the target node, you could run:
```
PROCESS WHERE (Name="cmd.exe") LIST BRIEF
```

In order to display processes which report highest WorkingSetSize (corresponding to the Memory
column in Task Manager), e.g. higher than 10 MB, you could run:
```
PROCESS WHERE (WorkingSetSize>10000000) LIST BRIEF
```

If you want to monitor these values over a longer period of time, you can use the /**EVERY** and /**RE**
switches. /**EVERY** sets the interval (in seconds) after which the list of processes is listed, while /**RE**
specifies the number of times that the interval is counted (and the current process statistics are
displayed).

For example, the following command will display processes with WorkingSetSize above 10 MB 3 ti
5 second intervals:
```
PROCESS WHERE (WorkingSetSize>10000000) LIST BRIEF /EVERY:5 /REPEAT:3
```

If you don't want to limit number of intervals, you can simply omit the /**REPEAT** switch.

If none of the display formatting options that **PROCESS LIST** command offers suits your needs, yo
also use **PROCESS GET**, which displays only the properties which you specify.

For example, the following command will display only Name, ProcessID, and WorkingSetSize for e
process.
```
PROCESS GET Name,ProcessID,WorkingSetSize
```

Keep in mind that by using the /**NODE** switch, you can collect the process information from any nu
computers.

Terminating processes

PROCESS DELETE - allows you to terminate processes.

For example, the following command will terminate EVERY instance of Notepad.exe running on th
node.
```
PROCESS WHERE (Name="notepad.exe") DELETE
```

If you want to affect only one particular instance, you should use ProcessID instead (which can be
by running the **PROCESS LIST** command). Assuming that the process instance has the ProcessI
1824, you would next issue the command:
```
PROCESS WHERE(ProcessID=1824) DELETE
```

Creating processes

Even though the list of verbs available with **PROCESS** alias includes **CREATE**, it seems that eithe
implementation or the documentation about it is incorrect (**PROCESS CREATE** method produces
error message). However, since WMIC is really not much more than a friendly interface to WMI, in

absence of a working WMIC method, we can turn to its less friendly WMI equivalent.

The following script allows you to invoke the Create method of the Win32_Process WMI class and effectively, launch a process on a local or remote machine (specified by setting the value of sComp variable). For the sake of simplicity, I omitted a few additional parameters that can be provided wh creating a process (such as process priority or window type).

If you save the script as CreateProcess.vbs, then, for example, launching notepad.exe on the targ computer called TestPC would require you to type the following at the Command Prompt:

```
cscript //nologo CreateProcess.vbs TestPC %windir%\system32\notepad.exe
temp%
```

Note that when launching a process on the local computer, you can simply type a dot (.) instead of computer name.

```
Dim sComputer            'computer name
Dim sCmdLine             'command line of the process
Dim sCurDir              'working directory of the process
Dim oProcess             'object representing the Win32_Process class
Dim oMethod              'object representing the Create method

Dim oInPar               'object representing input parameters of the me
Dim oOutPar              'object representing output parameters of the r

sComputer                = WScript.Arguments(0)
sCmdLine                 = WScript.Arguments(1)
sCurDir          = WScript.Arguments(2)

Set oProcess             = GetObject("winmgmts://" & sComputer & _
                           "/root/cimv2:Win32_Process")
Set oMethod              = oProcess.Methods_("Create")
Set oInPar               = oMethod.inParameters.SpawnInstance_()
oInPar.CommandLine       = sCmdLine
oInPar.CurrentDirectory = sCurDir
Set oOutPar              = oProcess.ExecMethod_("Create", oInPar)

If oOutPar.ReturnValue  = 0 Then
        WScript.Echo "Create process method completed successfully"
        WScript.Echo "New Process ID is " & oOutPar.ProcessId
Else
        WScript.Echo "Create process method failed"
End If
```