6

A Look Into the Future of WMI Scripting

Discussing the future of any technology is always a challenge, since so many things may change between the moment you write down where you think the technology will go and the time you see the final implementation. However, by looking at what we have today and the way Microsoft analyzes the feedback it has received from customers, we can arrive at some "best guesses."

The current scripting infrastructure, implemented by WSH 5.6, is powerful enough to perform most tasks required by administrators. Leveraging the WSH 5.6 infrastructure via various COM object models, such as ADSI and WMI, to gain access to the system settings and monitoring provides administrators with a powerful set of tools. However, learning these technologies is not particularly easy for administrators, especially for those who do not want to be programmers. As explained in the WMI examples in this book, you need to know, understand, and master many interfaces and other components to harness all the power offered by the WSH/WMI combination. More specifically, to play with WMI, you must learn:

- The CIM repository mechanisms, to understand their organization and how to get access to the classes representing real-world manageable entities.
- The WMI providers capabilities, since they determine the classes capabilities (i.e., update, enumeration, events).
- The classes contained in the CIM repository, since they represent the template of the real-world manageable entities.
- The WMI COM object model or the .NET Framework classes, to understand how to get access to the management data from a coded logic in a script or a C# program.

- A scripting or programming language (i.e., VBScript, JSCript, VB.NET, C#), to create the core logic of the tasks (i.e., structure, loops, statements, etc.).

If you look at these technologies from a high-level perspective, you realize that you must understand and master a large number of concepts and technologies to effectively script on top of WMI. Actually, this is the root of many of the new ideas, and where the foundation for the future of the Windows management layers may come from, since Microsoft wants to simplify the amount of knowledge required to work with the management data available on the Windows platforms.

6.1 The .NET Framework and scripting

Although the .NET Framework is a well-determined Microsoft strategy today and represents a step forward in simplification, you still need to understand some of the underlying mechanisms that WMI takes advantage of, such as the CIM repository and the nature of the classes (i.e., instance class, event class). The .NET Framework *System.Management* and *System.Management.Instrumentation* namespaces deliver a new management programming object model, which is more consistent than the isolated islands of interfaces implemented by COM technologies, such as is the case when WMI is combined with ADSI or CDOEXM to do some Exchange management. Unfortunately, we already know that the current .NET Framework classes do not represent all the manageable elements of a Windows platform, since most features rely on the existing COM plumbing. As such, there are still many interfaces that need to be provided before we will be able to write code to manage every facet of the Windows operating system. Keep in mind that "managing" implies the concepts of "configuration," "monitoring," and "alerting." In this sense, the current .NET Framework classes still have their limitations, since most information exposed by the .NET classes relies on WMI.

To improve matters, Microsoft is considering the development of a new Windows management layer, which will include the current WMI implementation, as well as some new implementations to cover areas not yet contained in WMI. Since the access method to the management data will be .NET based, the transparency of the access method will be handled behind the scenes by the .NET Framework abstraction layer. Regardless of how the management data is retrieved (via the existing WMI implementation or some new technologies still not fully defined), the Windows management layer will provide the information in an abstracted way. The goal of such an

implementation is to hide the complexity that you cannot skip today if you are willing to use WMI. The new abstraction layer, acting as a management infrastructure service, will ideally be more object oriented when compared with the current implementation, which requires too much effort to work with. Of course, one can say that the current implementation is already object oriented, since we deal with COM objects all the time! So, where is the difference? In an ideal object-oriented world, a developer doesn't normally deal with the infrastructure implementation as we do with WMI. The essence of objects available through the model should provide the necessary information without having to deal with its core implementation. From that perspective, WMI is far from an ideal object-oriented programming environment. This is a major focus for Microsoft as it drives for future simplification. Bear in mind that this vision is a generic representation of how things could be. There is no implementation available in any form at the time of this writing. In fact, these statements are purely the vision of things that may be realized over a two- to three-year time frame.

Another aspect is the scripting environment itself. Today, we have two directions clearly defined in Microsoft technologies: the programming model, based on the .NET Framework class model and the new line of .NET Server products such as Windows Server 2003, and a set of next-generation servers to replace current versions of products such as Exchange and SQL that will be built on the .NET framework. The question we must ask is: Where do scripting technologies fit?

The aim of scripting technologies is to fill a gap between the pure programming world only used by "real programmers" and the user interface mostly used by support teams, help desks, and administrators. However, most enterprise administrators need to automate tasks in their day-to-day work, but they are not programmers and don't want to be one. Even if the .NET Framework class model offers most of the features to create managed code and manage the Windows environment, it does not necessarily represent a viable solution for enterprise administrators. In fact, you very rarely find administrators who install **Visual Studio.NET** and start learning languages such as C# or VB.NET just to write a few lines of code to automate simple or repetitive management tasks. In addition, any good operating system always provides a shell with a scripting language to develop coded logic (i.e., the Korn and C shells for the UNIX platform or REXX for IBM systems). Therefore, Microsoft needs to fill a space in its operating system that offers the advantages of the scripting environments, as we know them today, and exploits the power of the .NET environment at the same time.

6.2 Windows scripting environments

Under the current Windows Operating System platform, we have two important scripting environments implemented by **CMD.EXE** and WSH. **CMD.EXE** uses the old-fashioned CMD (command-line) commands, while WSH is purely COM/DCOM based. When looking at the Microsoft .NET strategy, how do the CMD and WSH environments fit into the .NET Framework, which uses a completely different run-time environment? This current .NET Framework run-time environment is currently not designed to support scripts, or at least in a way we know scripting today. This is where Microsoft is still determining many options they can pursue over the next two to three years.

We have a good realization about many of the shortcomings of the existing scripting environment. For the CMD environment, limitations exist, such as not being able to access system management data without the use of an external executable or script that acts as a "black box." For WSH, we have another set of limitations, such as the use of non-strongly-typed languages such as VBScript and the inability to invoke any .NET Framework features or Win32 APIs (unless you use PERL with a third-party scripting engine that supports Win32 API calls). Another confusing problem is the need to use different COM technologies to retrieve management data (i.e., ADSI, CDOEXM, WMI, to name a few). In addition, these COM technologies sometimes overlap each other (i.e., you can start and stop Windows services with WMI, but you can do this as well from ADSI), which creates additional confusion.

Although there is no concrete architecture or components today to address these limitations and leverage the actual scripting technologies via the power of the .NET framework, Microsoft would like to eliminate most of the limitations that exist in the current environments while maintaining the advantages that the scripting technologies offer today. In this respect, the golden rule is "simplicity"! In other words, the technology must be easy to learn and use.

Looking into the future, we could predict that Microsoft will get rid of the current CMD shell, since it is the origin of many limitations. A new CMD shell built on top of the .NET framework could replace the current version. This development may also imply that we will see a new WSH environment. Currently, WSH is based on a COM/DCOM-based architecture, but we know that Microsoft is basing any new development on top of the .NET framework, so it is clear that the "new" WSH will certainly not be the same as today. It is more than likely that a new shell will support a more

powerful command-line language while supporting one or more scripting languages. It is also possible that the new shell will support both graphical and text interfaces to make script writing and execution easier. For instance, with WSH today, any administrator can start Notepad, write a few lines in Jscript or VBScript, and launch the script to begin executing. This is an ease of access to programmatic control that most programming languages don't support, and Microsoft is conscious of the advantages delivered by such an implementation. However, it will take time before we get to a Windows platform that is 100 percent .NET enabled, and, because of this, we can be almost certain that the current COM/DCOM WSH and WMI architectures will not vary enormously over the next two to three years.

6.3 Final thoughts

Of course, all these probable changes may discourage people from getting to know WSH and WMI in their current form, but you should always remember that the technology of today solves today's problems and the technology of tomorrow is not yet available. Since Windows Server 2003 (and before) is based on COM/DCOM, the current WSH and WMI implementation will remain in use for many years, which reinforces the importance of investing in scripting technologies and WMI today, even if Microsoft is considering where a new architecture for scripting might bring us in the future.