Realtime publishers

October 2008
Volume 1, Number 9

Windows Administration
in Realtime

Letter from the Editor

Automation Tools for the Small IT Shop

by Greg Shields

If you're an IT professional in an SMB environment, you might find yourself lusting after the automation tools the big enterprises seem to enjoy. The combination of tools like System Center Configuration Manager and System Center Operations Manager means that the big guys find out as problems occur and automatically push out solutions in the blink of an eye. But in your small IT shop, often with a zero- or negative-dollar budget, you're usually forced to use whatever duct tape and bailing twine you can scrounge from around the Internet's free download sources.

Here at Realtime, we hear your pain. We make an effort to talk about not only the tools that the deep pockets can afford but also those solutions that require nothing more than a little intestinal fortitude and whatever native tools are available with Windows itself.

That's why in this month's issue, I'm starting a two-part series on automating software deployment for the small IT shop. There is some secret knowledge hiding in the IT blogosphere that can help reduce your overloaded self. By automating the software installation nightmare, you immediately take a big all-manual pile of work off your plate. Plus, what you learn in the process takes you far into other aspects of automation, such as scripting and automated actions. All you need is the time and the inclination to do it.

So if you're a small IT shop, or even a big one that hasn't started automating, take a peek at my feature piece in this month's issue. You might find in there the information you need to get out of the office just a little earlier this month. As a fellow systems administrator and long-time friend of mine used to say, "A good systems administrator never works more than 40 hours a week."

For this crisp fall month and for the rest of the year, let's all make that our goal.

Answers from the Experts

Should you be looking at ILM?

by Don Jones

Q: David, a reader of my ConcentratedTech.com blog, asks, "I'm reading more and more about Microsoft Identity Lifecycle Manager (ILM). We currently use Active Directory—is ILM something I need to be looking at?"

A: David, ILM is one of those stealth products that seem to spring fully-grown from Redmond. ILM was actually a Microsoft acquisition, and was originally called Zoomit Metadirectory. If the word "metadirectory" is familiar to you, you know what ILM is all about. Basically, it's a super-directory. The idea is to manage all your users in ILM, and then let ILM "push" those users out to the various "real" directories that you own.

First, some terminology: A *directory* is any database of user accounts. *Identity* is just a fancy way of referring to user accounts: Accounts are computers' way of identifying human beings; therefore, accounts = identity. An identity's *lifecycle* covers its full life, from creation to modification to deletion. When a new employee joins the organization, you need to create accounts for that new person—usually in several places, such as Active Directory (AD), an Enterprise Resource Planning (ERP) system such as SAP, perhaps a human resources system such as PeopleSoft, and so on. As the person's role within the organization changes over the months and years, you need to modify that account, usually by assigning it new permissions or placing it into different groups. Finally, when the person leaves the organization, you need to remove, or *de-provision*, the account. ILM seeks to handle all of that. At its core, it's a big list of user accounts coupled with a bunch of synchronization connectors that know how to "talk" to various different directories—such as AD, PeopleSoft, and so forth.

The version of ILM under development—ILM "2" they're calling it, because apparently the ILM team wasn't issued any cool code-names like Longhorn—adds a bunch of important new features. It includes self-service capabilities that are integrated into Microsoft Office, and it adds a lot of change control and workflow capabilities so that identity management can be a part of broader, formal IT management processes (such as processes based on frameworks like COBIT or ITIL).

In theory, ILM helps centrally manage permissions, too, by placing users into groups. You then assign the right permissions to the right groups, and everyone's happy. In reality, Windows' management of resource security is awfully decentralized, making the process of assigning permissions to groups incredibly time-consuming and awkward—not to mention difficult to manage in the long term, as permission reporting doesn't exist natively in Windows and there are no real controls on who can change permissions over time. That's why vendors such as NetIQ, NetPro, and Quest all have popular third-party access management products—most of which are billed as integrating or complementing ILM. ILM handles the identity and putting users into groups; the third-party product manages the permissions that the groups themselves have on various resources (such as files, folders, registry keys, services, Exchange Server, AD, SQL Server, and the like).

When Microsoft bought Zoomit, the metadirectory world seemed to be on the verge of massive growth. Today, relatively few mature solutions continue to do well: ILM is one of them. IBM has Tivoli Directory Integrator, Novell makes Identity Manager, and there are other players—Sun, Oracle, and a few others. If you're primarily a Microsoft shop but you have systems whose directories don't integrate directly with AD, ILM is probably something you need to be aware of.

Do you have an IT question you'd like Don to answer? Send it to answers@realtimepublishers.com for consideration!

Don Jones is a co-founder of Concentrated Technology (www.concentratedtech.com), helping to deliver IT knowledge in less time using innovative content techniques. He also serves as CTO and Series Editor for Realtime Publishers. Don is the author of more than 30 IT books, including Windows PowerShell: TFM; VBScript, WMI, and ADSI Unleashed; Managing Windows with VBScript and WMI; and many more. He is a multiple-year recipient of Microsoft's "Most Valuable Professional" (MVP) Award with a specialization in Windows PowerShell.

Product Review

PowerShellPlus Professional Edition

by Don Jones

A couple of years ago, Microsoft MVP award winner Dr. Tobias Weltner released an early beta of what he called PowerShellIDE, a graphical development environment for Windows PowerShell. Building on his VBScript-centric SystemScripter product, PowerShellIDE offered a good editing experience for PowerShell users. At close to the same time, fellow MVP Karl Prosser was working on PowerShellAnalyzer, a product that offered a more interactive PowerShell experience through an alternate console host rather than an editor. Weltner and Prosser joined forces to create ShellTools LLC, which then began working on PowerShellPlus, which offers both an improved interactive console and a full IDE—sort of a combination between the two separate projects. Now, Idera, known for their data management tools focused on SQL Server and SharePoint, is working with Weltner to offer PowerShellPlus Professional Edition. As of this writing, the product is not generally available, but I was offered a first look at it. Since it's based on the already-released and stable PowerShellPlus, it's pretty ready for "prime time." Idera's worked with Weltner to make a bunch of usability changes, consolidating similar functionality into one place, and so forth—really just polishing the product very nicely. There's also a new Learning Center, which includes tutorials and other useful educational information.

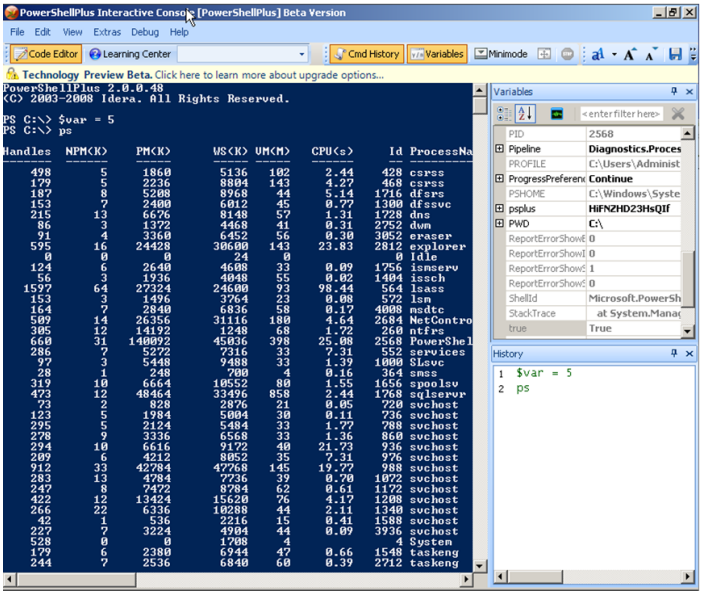

Figure 1: The interactive CLI.

As Figure 1 shows, the product starts with an interactive command-line interface (CLI) similar to PowerShell's native console. This new console, however, corrects the most egregious shortcomings of the native console, offering better options for font selection and size, improving Clipboard operations, and providing a set of panes (on the right) which perform helpful tasks like tracking your command history, displaying defined variables, and so forth.

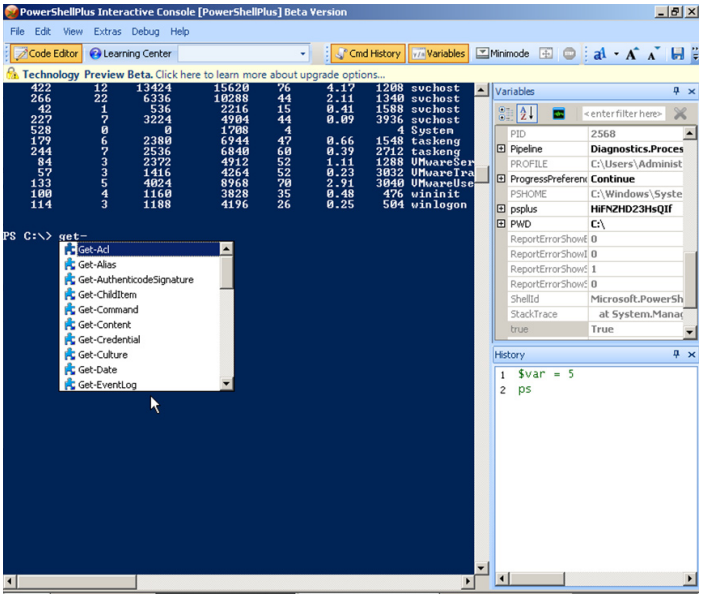

Figure 2: PowerShellPlus CLI provides features more commonly associated with an IDE.

Figure 2 shows how the PowerShellPlus CLI also provides features more commonly associated with an IDE, such as pop-up code completion menus when you're typing cmdlets. This code completion even extends to cmdlet parameters. A "mini mode" turns the CLI into a transparent, always-on-top resizable console window without any "chrome" (that is, without any toolbars, menus, etc.), making it easy to position anywhere on the screen and use alongside other applications.

A toolbar button opens the code editor, which offers the usual features such as code hinting, syntax coloring, and so forth. A great live syntax-checking feature (see Figure 3) calls your attention to typos and other errors that can be statically

detected by the PowerShell engine. It won't catch everything because some errors—especially logic errors—can only be found when your script is running. That's made easier by a built-in debugger, a piece of technology that had to have been difficult to piece together because PowerShell v1 doesn't offer any kind of real built-in debugging capability.

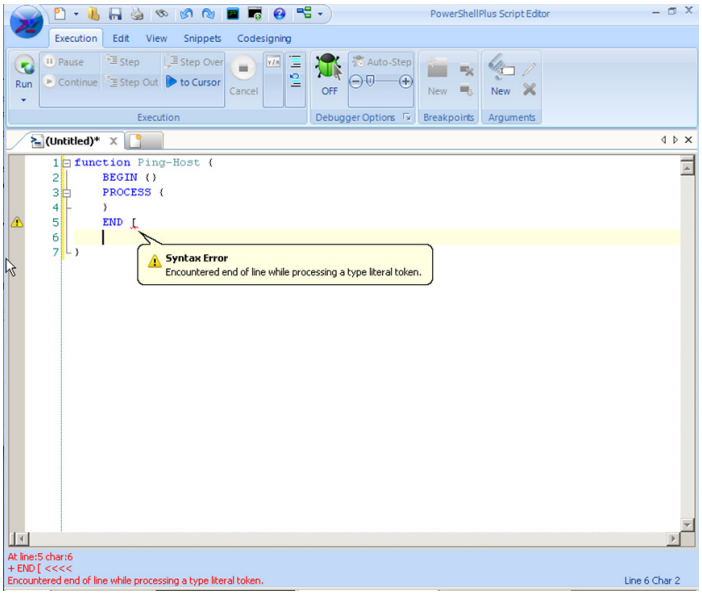

Figure 3: The product's live syntax-checking feature.

Figure 4 shows this debugger in action, illustrating how it displays variable contents when you hover your mouse over the variable—similar to the way professional IDEs such as Visual Studio behave. The debugger runs scripts pretty slowly compared with running them in PowerShell itself, but that's to be expected given the perambulations no doubt occurring behind the scenes to make this happen. There are a few "gotchas" at this point; the hover-over-a-variable trick, for example, doesn't work with the special $_ variable that contains the current pipeline object. Things like that can be worked around by—in this case, assigning the $_ variable's contents to a regular variable near the top of the script.

The editor includes Snippets, which are an invaluable way to re-use bits of code. It's also capable of signing your scripts so that you can have a more secure PowerShell scripting environment.

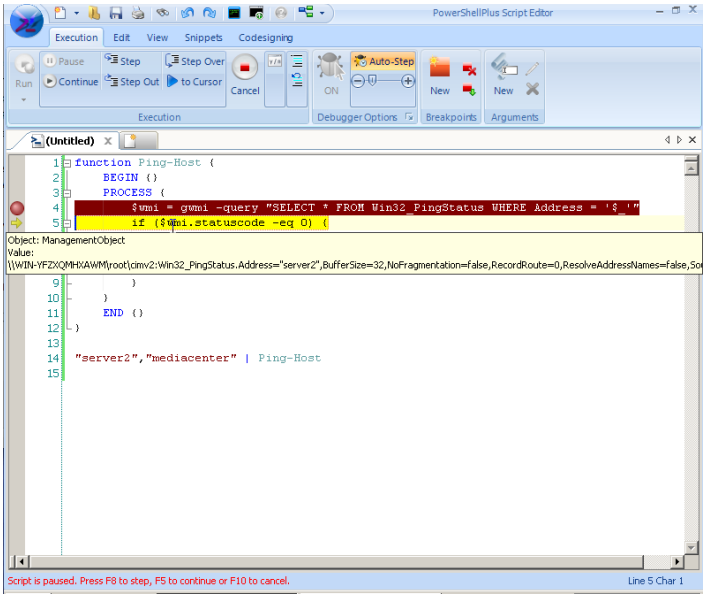

Figure 4: The product's debugger.

Overall, both the editor and console worked pretty flawlessly for me. A huge plus for me is that the program (apparently written in .NET Framework, which is fine because PowerShell itself requires the Framework) doesn't actually require an installation. It is shipped in an installer that includes several additional minor tools (such as the difficult-to-find MakeCert.exe tool), but the main executables can simply be placed onto a USB key. This makes the applications easy to carry to a server, run, and take away without leaving a footprint.

Idera is currently offering the product for review to industry insiders such as MVPs; by the time you read this, they'll be offering the product as a public beta, meaning you can get your hands on it and offer feedback. That period should last through most of September, at which point a commercial release will be available. Beta testers can pre-purchase it for just $80; final pricing will be $145 at Idera.com/purchase. Idera is officially the owner of PowerShellPlus from this point on; the product won't be offered by ShellTools, LLC any longer. Weltner will continue to develop the product in conjunction with Idera. Idera is looking to use this product to become a broader Windows management company, moving beyond their traditional SQL Server, SharePoint, and Microsoft Dynamics space.

Are You Ready?

Common Ways to Tell You Are Not Prepared to Recover from a Disaster

by Eric Beehler

Disaster recovery is somewhat of a buzzword in the IT industry, and IT professionals have all been exposed to their share of great disaster recovery ideas from business managers. These ideas are often based on the industry buzz and seem to only make more work for you with little gain overall. This is usually because the idea is not backed up with a real plan. The actual implementation of disaster recovery is usually a big chore to undertake correctly, but in the end, it is well worth the trouble.

It's important to be ready to recover your data and systems when a disaster strikes, but it is rarely a top priority in the grand scheme of IT projects when crisis has yet to strike close to home. Unless your company has decided to make disaster recovery a high-level objective, it's usually the front-line administrator that will be saddled with the responsibility of implementing some sort of plan to save the day -- but you will likely be short changed on training and resources to get the job done.

There are many ways to deal with a disaster, from having a set of cold standby machines to employing a fully redundant hot data center. In reality, as the administrator, your job doesn't change much based on the scenario for recovery; it has to be up and available to keep your business running. You likely have some kind of plan now, but if you haven't been through the real thing, you really don't know if your plan will hold water. For Windows administrators, there are several problems that seem to expose themselves when it's time to exercise a disaster recovery plan, or worse yet, go through the real thing. Here are some common ways to tell that you are not ready for a disaster.

Plan for an Alternative Site

You are not ready for a disaster if you don't have a place to go, which requires planning for a full on-site disaster in which your site is down or inaccessible. There are several methods to address this issue if you don't have a solution today, from having an alternative site with servers waiting to be loaded up for operation to a warm site that is always ready and waiting to take traffic. These decisions are not usually made by you but by the CIO. All you can often do is consider the solution given to you and how that will impact your ability to recover. A cold site, for example, will allow you to have hardware and connectivity available, but you will need to account for operating systems (OSs), drivers, configuration differences, and data center differences. In a warm site, you have to ensure that changes to configurations and data remain synched across the two sites.

Plan for Downtime

You also have to consider whether the site solution will support the Recovery Time Objective (RTO) required by the applications and business. Simply put, the RTO is the amount of time your users will be without the functions supported by your server, which could be a Web site, a mailbox, or the ability to log on to the domain. You should have this time defined per application or function supported by your server. This, of course, in a bigger effort for disaster recovery, may be defined for you, but don't be surprised if the business people you support have no idea that your server supports the functionality they require. You may need to interject with your personal knowledge of how your server functions in order to get this definition correct.

There are generally accepted categories for RTO that fall into tiers, as Figure 1 shows. Use these as a guideline but feel free to create standards within your own organization to meet your needs. If you have a need to recover applications with 2, 4, and 8 hours, redefine the tiers so that they make sense to your business through an analysis of the business impact of downtime. Just be sure that you can apply the standards as broadly as possible across the organization.

RTO Tier	Definition
Tier 1	Fault tolerant with no appreciable impact to the end user if a system goes down.
Tier 2	Unavailable less than 24 hours
Tier 3	Unavailable less than 48 hours
Tier 4	Unavailable between 2 and 7 days
Tier 5	Unavailable more than 7 days

Figure 1: Example RTO tiers

Plan Your Tolerance to Data Loss

You are not ready for a disaster if you don't know your tolerance for data loss. Let's start with the basic foundation of the backup. Whether you use simple tape backups or an advanced nearline solution, you have to consider that most solutions are put in place to account for day-to-day operational needs. First, the exercise you went through with RTO must be done for the Recovery Point Objective (RPO), which is the amount of data that can be lost. You have to understand what the business can afford to lose; this value is not necessarily tied to an RTO tier. Take, for example, a point of sale system. If the system is down for 5 hours, the business may be able to recover by entering the orders taken while the system was down, but data loss of 5 hours may mean millions of dollars in lost sales.

RPO Tie	Definition
Tier A	No data lost
Tier B	Less than 24 hours of data loss
Tier C	Data loss up to the last backup (usually 24 to 36 hours of data loss)

Figure 2: Example RPO tiers.

The gut reaction for your RPO on some of your systems may be that no data loss is acceptable. In other cases, 24 hours of data loss may be acceptable. The goal is to understand what can be tolerated, not what is desired. Everyone will desire no data loss, but put a realistic perspective to the real value of the data. If you define Tier A RPO as no data loss, then you have to put systems in place that allow for that reliability. This means copying transactions as they happen to a backup site, which is an expensive solution that should be used only on your critical business applications, depending on your budget. If you have Tier B systems as defined in Figure 2, you will need some sort of solution that will be separate from your nightly backups, as you cannot count on having your last nightly tape backup at your recovery site.

Considering the Loss of a Backup

You are not ready for disaster if you rely on your daily backup for a recovery scenario. You may have in your head that you can rely on the last tape backup in the event of a disaster. Whether such is the case depends on a key question: can you get your restore process to work offsite? Don't be so quick to answer this one. If you take advantage of offsite storage either through a vendor or your own in-house process, it is an excellent step, but offsite storage doesn't necessarily guarantee you can restore at your disaster recovery site within the specified RPO and RTO. Tape drive compatibility, backup software, delivery time, drivers, and OSs are all considerations that you must address prior to saying your solution is ready. This is especially true for a third-party backup site that will provide you with "like" hardware. That equipment will not be your equipment, and even if it is, expect aspects of the infrastructure to be different, such as IP address schemes, firmware (which can be a nightmare when working with SANs), and simple access to the hardware.

You also have the issue of archive requirements and the fact that you likely rely on these tapes for your day-to-day restores. If you perform restores for file recovery and other issues, you likely want to keep those tapes close by. If you ship them away for maximum protection, it's going to cost a pretty penny in order to request tapes from your offsite storage vendor.

You also have to consider how those tapes make it to the recovery site. If you make full backups only once a week and you only do offsite storage once a week, you might only get a restore from 2 weeks prior. Why? Because if you are lucky enough to get your tapes offsite a day or two after the full backup and you get the shipment to your disaster recovery site 4 to 8 hours after they are requested, you can almost bet that Murphy's Law will strike and you will get a bad tape somewhere in the set. Then you have to move back in the chain, and with most full backups run weekly, you might be taking you system back 2 weeks or more if Murphy continues to strike. Now, the RPO of your plan that you expected to meet with your existing backup plan is not being met.

Even if you do recover your servers with no issues, how long will it take to recover them all? Consider the queuing on the tape drives, with multiple servers waiting for those tapes to be loaded. It could take quite a long time before you even get a chance to try a restore to your server depending on the technology present at the recovery site. What can you do? Well, time to restore will be reduced if you can restore large chunks at one time. Consider putting systems with like RPO and RTO requirements in the same backup set. Better yet, host them on a LUN or set of LUNs on your SAN or other logical storage method in your situation so

that a restore can be done all at once. You might even consider booting from the SAN, which might save you from having to restore the local disk of many servers. If you have a blade server solution, this may even be baked into your infrastructure.

Using Disk-Based Backup

Let's also consider disk-based backup. This solution has become increasingly popular because of the low cost of hard disks and the ease of backup and restore. In addition, disks often take minutes to back up and restore what used to take hours. The software supported by these systems even has versioning, much more frequent backups, and nifty utilities that make life much easier on the administrator. This is usually all handled by complex backup management software such as Microsoft System Center Data Protection Manager. When using this kind of solution, consider employing these often-integrated features to support data replication of some sort, although vendors name these types of features differently. You can even copy your live data to your recovery site using a SAN/NAS vendor's Failure Resistant Disk Solution (FRDS). You should, however, consider the fact that this kind of solution will be much more expensive than tapes because it will require duplicate equipment with data replication happening across a wide area network (WAN). You should refer to your RTO and RPO tiers to determine whether certain servers and data sets could stand to be away from your disk replication and rely more on a tape solution. You should also consider your disaster site and understand whether it can support this kind of solution. You should treat your server restores as a form of triage. You need to know, based upon RTO and RPO, what you are going to recover first and what can wait.

Considering Configuration

If you can't identify the full configuration of your servers, you are not ready for a disaster. Realistically, can you keep track of 300 shares on a terabyte SAN served by a load-balanced Windows cluster server? Do you know which shares go to which directories on which LUNs? You have to document configurations. This is true whether you have a basic bare metal restore plan or a full redundant data center. The luxuries of a production environment won't be at your disposal. A normal production environment allows you the opportunity to compare configurations when something goes wrong and work through a problem. A disaster affords you no such luxury.

No matter how familiar you are with your systems, you need to have everything documented that can be changed. For any applications, you should have a guide for their installation in your environment. You should have the servers documented with everything from IP addresses and patches to database connections and configuration files. If you run IIS for Web applications, you should have that configuration documented as well. Some sort of context diagram is often useful to determine how your server interacts with other systems.

Utilize configuration management systems, such as SMS, to do some of the heavy lifting for you. Create reports and keep them up to date in an alternative location, either a paper copy offsite or an electronic one. Configuration problems seem to be a killer when recovering because changes sometimes get applied without strict control. What seems like a small change can kill you in a disaster when it hasn't been documented.

Documenting the infrastructure goes beyond your own servers, but is just as important when it's time to troubleshoot. You can bring your file server back and you can bring your application servers back, but if you don't have proper DNS or connectivity, no one will be connecting to those systems you've recovered. If you have dependencies on other systems, you need to identify them. Know what names should be in DNS, what IP addresses and subnet you are on, what systems you interact with such as database servers or other back-end services such as the DMZ or Internet access. When you tell a database administrator that your application is taking SQL errors, you should know what database server, database, port, connection type, and authentication type you are using. You should also know the user name and password being used, if there is one. Does the server break down into pieces? Does it have multiple applications or functions? Document those functions separately.

You can't think of server as a single system if your customers don't see it as a single function. Remember that restoring an infrastructure is many pieces to a whole, and you should not expect any of those pieces to work correctly as you can in a production environment. In fact, when you face an issue in production, it usually has a single root cause, but a disaster recovery will usually experience several major issues at the same time. You need to know where you stand in the ecosystem of your environment to understand how to identify and help fix those issues.

Identifying Single Points of Failure

If you have a single point of failure, you are not ready for a disaster. A single point of failure can ruin your nicely laid out plans. Although not a requirement for a disaster recovery, the 'N + 1' definition used when considering disaster recovery is many components backed up by a single component. You can still run into problems using N + 1, especially at a cold site where you have not been exercising your disaster recovery equipment to ensure its health. You might consider having additional servers of a similar capacity available above the minimum number required to recover just in case you experience a failure at your recovery site. An optimal solution will have redundancy built-in to your recovery site the way you have it outfitted at your production site. If you have a failover cluster in one location, you would do the same in the recovery site, even though you could technically get by with a single server, assuming that server functions as expected. You should also consider the interdependencies of your infrastructure, such as network, when you think of this issue. Single switches, routers, domain controllers, and sources of power can also be points of failure.

Single point of failure doesn't stop at the system level. You might have that one guy or gal who knows everything about your environment. When you're at his desk and something goes wrong with the system or a specific application, he always has the answer. This gal is a good person, but when it comes down to it, you can't rely on a single person. When a disaster strikes, the go-to person may not be available during the recovery phase—yourself included. When everyone looks around and throws up their hands because such and such is down, what do you do? You wish you could go back in time and document that ingrained knowledge. This is also true for day-to-day operations, but especially necessary when everything is going wrong because of a disaster. The person who knows it all is not what you need, you need full documentation of the knowledge that person possesses. Your go-to should really be your documentation.

Integrating Disaster Recovery into Daily Life

If you don't integrate disaster recovery into your daily operations, you are not ready for a disaster. Organizations that plan for disaster recovery as a single project with a start and an end will fail. Don't let the hard work go to waste. When you put these plans in motion, get all that documentation done, have recovery solutions in place, and continue to update your documentation and test your systems. If you don't test you disaster recovery process regularly, how do you know it will work? If you don't update your documentation day-to-day when changes are made, your documentation is outdated and may even be detrimental to your recovery efforts. Don't let apathy or a disconnected process of change management get you in the end. Not only does integration help your readiness, it reduces the dedicated time necessary to getting disaster recovery ready. Find a way to make what you use in disaster recovery a part of daily life.

Eric Beehler has been working in the IT industry since the mid-90's and has been playing with computer technology well before that. From Help desk technician to solutions provider, he has been involved at many layers of enterprise solutions from the desktop to the network to the server and the SAN. He currently has certifications from CompTIA (A+, N+, Server+), and Microsoft (MCITP: Enterprise Support Technician and Consumer Support Technician, MCTS: Windows Vista Configuration, MCDBA SQL Server 2000, MCSE+I Windows NT 4.0, MCSE Windows 2000, and MCSE Windows 2003). He also holds a Master's degree in Business Administration from the University of Colorado at Colorado Springs. His experience includes more than nine years with Hewlett-Packard's Managed Services division, working with Fortune 500 companies to deliver network and server solutions and, most recently, IT experience in the insurance industry working on highly available solutions and disaster recovery. He has co-authored books, including MCITP: Microsoft Windows Vista Desktop Support Enterprise Study Guide (Sybex/Wiley Publishing), authored several white papers, and co-hosts the "CS Techcast" podcast aimed at IT professionals. He provides consulting and training through Consortio Services, LLC.

The Deep Dive

Automating Software Deployment for the Small IT Shop, Part 1

by Greg Shields

"Next, Next, Finish." Hop over to the next desk. "Next, Next, Finish." Another hop to a third cubicle. "Next, Next, Finish." Lather, rinse, repeat.

If you're the IT professional in a small environment, you're probably familiar with this mind-numbing process. Getting software installed to computers across your environment is a maddeningly repetitive series of button clicking and "Are you sure?" responses. After a few dozen or hundred of these, you may be saying to yourself, "There's got to be a better way to do this."

There is, though you may have found that most of the tools and technologies for doing so are designed with the enterprise IT organization in mind. If you're a small IT shop with a few dozen or hundred machines and a tight budget, enterprise-scale tools are usually far outside your reach. You don't need comprehensive solutions and you might not even need ones that work 100% of the time. What you're looking for is a simple way to rapidly deploy new software and patches all across your environment without the need for hopping from machine to machine.

First and foremost, the complexities of automated software deployment can be a bit overwhelming for the newcomer. With little commonality between most software packages and their installation routines, simply figuring out how to do it can be difficult. But there are steps that you can follow to ease some of this process. Interested? Read on.

Running Silently

In this, the first of a two-part series on automating software installation in the small IT shop, we first need to discuss the process of *repackaging* a software installation. This process takes a standard "Next, Next, Finish" installation and reconfigures it to operate "silently." Software installations that have been reconfigured to run silent can then complete their tasks without any input from a user at the console of the computer. Once a software installation is repackaged to run silently, any tool that can remotely launch a process can be used to remotely deploy that software.

> In next month's eJournal, part 2 of this series will discuss no-cost and low-cost tools that you can use to actually deploy the software once repackaged.

There are three common ways in which software is typically installed to a computer:

- *MSI-based installations.* These installations, all of which have an .MSI extension, leverage the built-in Windows Installer Service to complete their task. They share this commonality, so they tend to be the easiest to repackage.
- *EXE-based installations.* A software installation with an .EXE extension typically uses its own built-in mechanism for installing. With a set of potential tools to create these files, there are an equal set of ways to silence them. With these, you'll find yourself needing a little sleuthing to delve their secrets for silence.
- *Differential-based installations.* When neither of the previous two installation mechanisms work for an installation, tools are available that can snapshot the configuration of a computer before and after an installation to determine what files and registry keys changed.

For the first two options, the solution for repackaging is in finding their "silent switches." These switches, such as */v /qn*, for example, instruct either the built-in installation code or the Windows Installer Service to install the package without prompting the user. For the third, special tools are required. We'll discuss all three in the sections that follow.

MSI-Based Installations

As discussed earlier, MSI installations are generally the easiest to repackage so that they run silently. With the Windows Installer Service being the point of commonality between all MSI packages, all packages tend to have a similar structure for which silent switches need to be used to run the package silently.

Generally, all MSI-based installations use the *msiexec.exe* tool to invoke their installation. The general syntax looks a bit like this:

```
msiexec.exe /qb- /l* {logfile.txt} /i {setup.msi} {NAME=Value}
```

In this code, each switch instructs the Windows Installer Service to accomplish a different task associated with the installation. Table 1 explains what each switch does.

Switch	Description
msiexec.exe	Invoke the Windows Installer Service
/qb-	Use a basic user interface but with no (modal) dialog boxes
/l* {logfile.txt}	Log all information about the installation to logfile.txt
/i {setup.msi}	Install the setup.msi application (as opposed to repairing or uninstalling it, which use different switches)
{NAME=Value}	[Optional] Set the NAME setting to the configured Value

Table 1: A listing of common switches associated with an msiexec.exe-launched installation.

The last item in the table is optional and requires additional explanation. For MSI-based installations, customizations for the installed software are stored in a database-like format. This means that settings that customize the installation for your environment—such as installation location, post-installation reboot suppressing, or other elements—can be set at the time of installation. With MSI installs, these are typically completed in one of two ways: either through setting individual values at the command line or by using a transform file. Transform files are used when the number of customizations is large, as the individual customizations are wrapped into a single file.

For example, to install Macromedia Dreamweaver v8, you could use the command:

```
msiexec /qb- /l* logfile.txt /i macromedia-dreamweaver-8.msi SERIALNUMBER={value}
REBOOT=SUPRESS
```

In this sample, the *macromedia-dreamweaver-8.msi* file is launched with two customizations, inputting the serial number at the time of installation and instructing the installer not to reboot the machine after the installation.

If a transform file were available for this same installation, it would change the installation to resemble the following:

```
msiexec /qb- /l* logfile.txt /i macromedia-dreamweaver-8.msi TRANSFORMS={transform.mst}
```

The hardest part about repackaging MSI files can be in finding the customization settings that can be set at the command line. MSI interrogation tools are available to do this, and some application vendors provide tools for generating your own transform files. The "art" in this process is in determining what they are and how to use them.

> One of the best ways of finding this information is to check out www.appdeploy.com. This location includes a clearinghouse of many common installations along with their customization options.

EXE-Based Installations

EXE-based installs can sometimes be more difficult than MSI-based installations because each EXE-based install has its own built-in mechanisms for repackaging for silent installation. Sleuthing to find the appropriate switches is much of the "art" of software packaging. The easiest place to start is by simply attempting to run the software installation with the */?* switch. This switch—as well as */help, –help,* and others—can often display a dialog box that presents the proper switches to be used for silent installation.

Other common switches that are known to work are */s* and */s /v/qb*. These switches are used by some of the common enterprise packaging solutions for silent installation.

Yet another solution commonly used is wrapping an MSI installation into an EXE file. When launched, the EXE file then launches the MSI installation bundled within itself. With these sorts of installations, the use of the /a switch can sometimes assist with extracting the MSI file from its host EXE.

The process works a bit like this: From a command prompt, run setup.exe */a*. This launches what is called an *administrative installation* of the software. Often, this administrative installation can prompt you through the installation as if you were installing it on a computer. Instead of actually installing the package, the process results in an MSI file preconfigured with your stated customizations. That MSI file can then be launched silently using the techniques discussed earlier.

> There is no common schema among EXE-based installations, so other switches can also be configured to run the package silently, such as */q:a /r:n, /silent, /passive, /quiet*. The clearinghouse at www.appdeploy.com also provides information about EXE packages.

Differential-Based Installations

Lastly, is the situation where no matter of sleuthing can determine how to directly run the software's installation file in silent mode. Often, such is the case when the software's developer has neglected to include the necessary coding to make it run silently. In these cases, the optimal solution for repackaging this software is through a *differential-based installation*.

In a "diff," a special piece of software is used that snapshots a barebones system. This system should include the same operating system (OS) and the minimum amount of software necessary to install the software package you intend to repackage. The snapshot captures each file, folder, and registry key present on that system. Once the snapshot has completed, a process that can take a few minutes, the software is then installed to the reference computer.

> At its core, a software installation is little more than a process that copies a set of files and folders to a target system and adds, updates, or removes a set of registry keys. Professional installers include additional functionality that streamlines this process, but in the background, these are the main steps used to install virtually all pieces of software.

Once the software installation is complete, the "diff" tool then re-scans the system to look for changes in the file, folder, and registry composition. Any changes are typically presented through a tree-like interface that allows you to selectively remove any extraneous findings (which are common). Once removed, the results are then repackaged into a new MSI file that is automatically configured to run silently.

The issue with this process is that most "diff" tools are exceptionally expensive and a part of enterprise-class software distribution platforms. These multi-hundred and often multi-thousand dollar software packages can be too expensive for the small IT shop. One long-standing and no-cost solution still available today is the software tool called WinINSTALL LE 2003. This tool should be installed onto a clean reference computer. Once installed, run WinINSTALL LE's Discover menu item to begin the snapshot/installation/re-snapshot process.

> You can find a copy of WinINSTALL LE 2003 at http://www.appdeploy.com/downloads/detail.asp?id=4654.

Though the information here only scrapes the surface of what's available in the world of software packaging, it serves as a starting point for getting you away from desk-to-desk roaming and manual installations. Next month, we'll take what we've learned here and plug the results into some commonly available solutions for deploying that software package.

Greg Shields, MCSE: Security, CCEA, is an independent author, speaker, and consultant, based in Denver, Colorado. With more than 10 years of experience in information technology, Greg has developed extensive experience in systems administration, engineering, and architecture. Greg is a contributing editor for both Redmond magazine and MCPmag.com, authoring two regular columns along with numerous feature articles, webcasts, and white papers. He is also the resident editor for Realtime Publishers' Windows Server Community at www.realtime-windowsserver.com. Greg is currently finishing his new book Windows 2008: What's New, What's Changed through SAPIEN Press.

Practical PowerShell

HotFixes! Get Your Red HotFixes!

by Jeffery Hicks

Keeping servers and desktops up to date and patched seems like an endless chore. Getting computers updated isn't necessarily that difficult. But what about finding what has already been installed? Using Windows PowerShell and Windows Management Instrumentation (WMI), you can easily acquire this information. What's more, you can work with objects that make reporting very simple by using cmdlets such as **Out-File** and **ConvertTo-HTML**.

In Microsoft-speak, any patch or update is referred to as a *hotfix*. These updates are tracked in WMI using the Win32_QuickFixEngineering class. Querying this class in PowerShell is as easy as:

```
PS C:\> get-wmiobject win32_quickfixengineering
```

The class has several properties, some of which may be populated depending on the computer, operating system (OS), or hotfix. The more you work with this clas s, you'll discover a few quirks that are OS dependent. For example, the *InstalledOn* property should display the date when a hotfix was installed. On Windows XP or Windows Server 2003, this property value is very recognizable as a date. However, in Windows Vista, the date format is less user-friendly—do you know what date this is: 01c8e2c0a6cfee4e? I certainly didn't.

To make your job a little easier and to take advantage of PowerShell's pipeline, I put together a script, Get-HotFix.ps1, which you can download from the Realtime site.

You can download the following code from http://www.realtime-windowsserver.com/code/v1n9_Practical_PowerShell.zip

```
#Get-HotFix.ps1
Param([string]$computername=$env:computername,
      [System.Management.Automation.PSCredential]$credential)

#function to return int64 UTC dates from Vista in a user friendly
#format
    Function Parse-UTC64 {
     Param([string]$value=0)

    #Sample $value='01c7e70de43f747f'
     $value=[system.Int64]::Parse($value,[System.Globalization.NumberStyles]::AllowHexSpe
cifier)

     [system.DateTime]::FromFileTimeUtc($value)

    }
```

```
#set to Continue if you want to see Warning messages
$WarningPreference=”SilentlyContinue”

#Set to Continue if you want to see Debug messages
$DebugPreference=”SilentlyContinue”

#Set to Continue if you want to disable the
#function’s error handling
$errorActionPreference=”SilentlyContinue”

Write-Debug “Starting Get-Hotfix function”

If ($credential) {
    Write-Debug (“Using alternate credentials:{0}” -f $credential.username)
 }

#my error handling traps
Trap {
    if ($_.Exception -match “RPC server is unavailable”) {
        Write-Warning “$computername is not available via RPC.”
        continue
    }

    elseif ($_.Exception -match “access is denied”) {
        Write-Warning “Access denied to $computername.”
        continue
    }
    else {
        Write-Warning “There was an error”
        Write-Warning $_
        continue
    }
}

if ($qfe) {
    Write-Debug “Removing leftover `$qfe variable”
    Remove-Variable qfe
}
Write-Host “Querying hotfixes on $computername” -foregroundcolor Cyan

$cmd=”Get-WmiObject -class win32_quickfixengineering -computername $Computername
-ErrorAction ‘Stop’”
```

```
if ($credential) {
    $cmd=$cmd + " -credential `$credential"
 }

Write-Debug "Scanning $computername"
Write-Debug $cmd

$qfe=Invoke-Expression $cmd

if ($qfe.count -eq 0) {
    Write-Debug "No quick fix engineering patches found on $computername."
    Write-Warning "No quick fix engineering patches found on $computername."

}
else {
    Write-Debug ("Found {0} hotfixes on {1}" -f $qfe.count,$Computername)

    foreach ($hotfix in $qfe) {
        Write-Debug ("ID {0}" -f $hotfix.HotFixID)

        if ($hotfix.Installedby -match "S-") {
            $sid=$hotfix.Installedby
            Write-Debug "converting $sid to username"
            #WMI should use cached connection, even if alternate credentials were used
            $user=Get-WmiObject win32_useraccount -computername $Computername -filter
"SID='$sid'"

            if ($user) {
                $installedBy="{0} [{1}]" -f $user.caption,$user.FullName
            }
            else {
                $installedBy="Not found"
            }
            Write-Debug $installedBy

            #overwrite existing existing property
            $hotfix | Add-Member NoteProperty -name "Installedby" -value $Installedby
-force

            }
```

```
 #if queried system is Vista convert InstalledOn to a “real” date

        if ($hotfix.InstalledOn.length -eq 16) {
            Write-Debug (“Converting {0} to a date” -f $hotfix.InstalledOn)
            $InstalledOn=Parse-UTC64 $hotfix.InstalledOn
        }
        else {
       #convert existing value to a date if it is something like
            #20080405
            if ($hotfix.installedOn -match “\d{8}”) {
            Write-Debug (“Parsing {0}” -f $hotfix.InstalledOn)
              $i=$hotfix.installedOn
              [datetime]$InstalledOn=”{0}/{1}/{2}” -f $i.substring(0,4),$i.
substring(4,2),$i.substring(6,2)
              }
            else {
            #leave it alone but make it a datetime if it exists
                if ($hotfix.InstalledOn -match “/”) {
                    Write-Debug (“Setting {0} to [datetime]” -f $hotfix.InstalledOn)
                    [datetime]$InstalledOn=$hotfix.InstalledOn
                 }
                 else {
                    Write-Debug “InstalledOn is most likely empty”
                    [datetime]$InstalledOn=”1/1/1970”
                 }
            }
        }

        $hotfix | Add-Member NoteProperty -name “InstalledOn” -value $InstalledOn -force
-passthru
        write $hotfix
      #remove variables to avoid any surprises next time through
      #the loop
        “User”,”sid”,”installedby,installedOn” | foreach {
            Remove-Variable $_ | Out-Null
        }

    } #end Foreach hotfix
  } #end else

  Write-Debug “Ending Get-Hotfix “
 #end script
```

Listing 1: Get-HotFix.ps1.

Usage is pretty simple. All you need to do is specify a computer name and an optional PSCredential. The default computer name is the local computer, so all you really have to do is run

```
PS C:\> c:\scripts\get-hotfix
```

To connect to a remote machine, simply pass a computer name:

```
PS C:\> c:\scripts\get-hotfix file09
```

To use alternate credentials, you can nest a **Get-Credential** command:

```
PS C:\> c:\scripts\posh\get-hotfix file09 (get-credential mydomain\administrator)
```

Or I prefer to create a saved credential I can reuse:

```
PS C:\> $admin=get-credential mydomain\administrator
PS C:\> c:\scripts\posh\get-hotfix file09 $admin
```

Let me give you a quick rundown on how the script works, then I'll show you some ways to use it. At the beginning of the script, you'll find variables to control PowerShell behavior as well as my error handling traps. Those should be self-explanatory. But when you get to this line:

```
$cmd="Get-WmiObject -class win32_quickfixengineering -computername $Computername
-ErrorAction 'Stop'"
```

You might be a little confused. I'll explain. Because you might specify an alternate credential, the script would need a different **Get-WMIObject** syntax. What I'm doing in my script is to build a **Get-WMIObject** command string. The $cmd variable is my basic command. If an alternate credential is passed, the command string is modified to include it:

```
if ($credential) {
    $cmd=$cmd + " -credential `$credential"
 }
```

I execute the command using the **Invoke-Expression** cmdlet and save the results to a variable, $qfe:

```
$qfe=Invoke-Expression $cmd
```

Although it is highly unlikely, there is a possibility that no hotfixes are installed. If that is the case, I want to display a message and not do anything else:

```
if ($qfe.count -eq 0) {
    Write-Debug "No quick fix engineering patches found on $computername."
    Write-Warning "No quick fix engineering patches found on $computername."

}
```

Otherwise, I display the number of detected hotfixes and start working with each one:

```
    Write-Debug ("Found {0} hotfixes on {1}" -f $qfe.count,$Computername)

    foreach ($hotfix in $qfe) {
```

Most of the hotfix object's properties are self-explanatory and fine as they are. However, depending on the OS, some of these properties need a little help. For example, the *InstalledBy* property should show you who installed the hotfix. On most systems, this should be a username, but on Vista, it might be listed as a SID. If that is the case, I've added code to convert the SID to a username using the Win32_UserAccount.

```
        if ($hotfix.Installedby -match "S-") {
            $sid=$hotfix.Installedby
            Write-Debug "converting $sid to username"
            #WMI should use cached connection, even if alternate credentials were used
            $user=Get-WmiObject win32_useraccount -computername $Computername -filter
"SID='$sid'"

            if ($user) {
                $installedBy="{0} [{1}]" -f $user.caption,$user.FullName
            }
            else {
                $installedBy="Not found"
            }
```

Once I have a new value, I overwrite the existing *InstalledBy* property by piping the current hotfix object to **Add-Member** using the –force parameter to force overwriting the existing property:

```
            $hotfix | Add-Member NoteProperty -name “Installedby” -value $Installedby
-force
```

Another property that may need some tweaking is *InstalledOn*. On older systems, this property value will be something like 7/4/2007. However, on Windows Vista, this value is written as a string like this 01c7e70de43f747f, which is hardly user friendly. The script takes this value if found and passes it to a function called Parse-UTC64.

```
        if ($hotfix.InstalledOn.length -eq 16) {
            Write-Debug (“Converting {0} to a date” -f $hotfix.InstalledOn)
            $InstalledOn=Parse-UTC64 $hotfix.InstalledOn
        }
```

The function parses value into a UTC string and then converts it to a more practical date time format:

```
$value=[system.Int64]::Parse($value,[System.Globalization.NumberStyles]::AllowHexSpecifi
er)

     [system.DateTime]::FromFileTimeUtc($value)
```

Some *InstalledOn* values may be written as 20080405, but this is a string. Because I’d like to sort or filter on this property as a date time, I hack the value to turn it into a string that PowerShell can convert to a [datetime] object.

```
            if ($hotfix.installedOn -match “\d{8}”) {
            Write-Debug (“Parsing {0}” -f $hotfix.InstalledOn)
              $i=$hotfix.installedOn
              [datetime]$InstalledOn=”{0}/{1}/{2}” -f $i.substring(0,4),$i.
substring(4,2),$i.substring(6,2)
              }
```

If the *InstalledOn* property contains a string 08/01/2008, then it too is changed into a [datetime] object:

```
                if ($hotfix.InstalledOn -match “/”) {
                    Write-Debug (“Setting {0} to [datetime]” -f $hotfix.InstalledOn)
                    [datetime]$InstalledOn=$hotfix.InstalledOn
                 }
```

Finally, it’s possible that the *InstalledOn* value will be empty. In that case, I decided to give it an arbitrary date of 1/1/1970 so that sorting results by this property won’t fail:

```
                    [datetime]$InstalledOn=”1/1/1970”
```

Again, I update the *InstalledOn* property with a newer value using **Add-Member**:

```
        $hotfix | Add-Member NoteProperty -name “InstalledOn” -value $InstalledOn -force
-passthru
```

Once the object has been tweaked, it is written to the pipeline:

```
        write $hotfix
```

The entire process repeats for the next hotfix in $qfe.

Here are some one-line examples you can try, which should give you an idea of how to use this script.

```
.\Get-HotFix | Format-Table HotFixID,Caption,Description,InstalledOn -auto

.\Get-HotFix SERVER02 | Sort Description | Format-Table -groupby
Description,HotFixID,Caption -auto
```

If you want to use alternate credentials, you'll find it easiest to use a stored PSCredential object:

```
$cred=get-credential “mydomain\administrator”
.\Get-HotFix DESK23 $cred | select Description,Caption,HotfixID,Install* | export-csv c:\
reports\desk23qfe.csv -notypeInformation.
```

More than likely, you'll want to check several computers. This is easily handled by parsing a text list using **Get-Content** and piping it to **ForEach**, which passes the computer name to the Get-Hotfix script:

```
Get-Content (“computers.txt”) | ForEach {
 .\get-hotfix $_ } | sort CSName,InstalledOn |
 Format-Table -groupby CSName InstalledOn,HotFixID,Description,Installedby |
 Out-File c:\reports\hotfixes.txt
```

The script's output is sorted, formatted as a table grouped by computer name, and then written to a text file. Or perhaps you might like to create an HTML report:

```
Get-Content (“computers.txt”) | ForEach {
.\get-hotfix $_} | sort InstalledOn |
Select CSNAME,InstalledOn,Installedby,HotfixID,Description |
ConvertTo-Html -title “QFE Report” | out-file c:\reports\qfe.html
```

In fact, let me show you how to take this example a step further. The hotfix object's *Caption* property is a string formatted as a URL. Wouldn't it be nice to make that a working link to the Microsoft support article in the HTML file? Here's how I do it. First, I'm going to need a regular expression pattern that will match on the URL:

```
[regex]$regex="http:\/\/support\.microsoft\.com+([\w\-\.,@?^=%&amp;:/~\+#]*[\w\-
\@?^=%&amp;/~\+#])?"
```

Now all I need to do is find all matching strings in my HTML file and replace them with a link tag. I'm going to take the existing HTML file and revise it, so first I read its contents into a variable:

```
$tmp=Get-Content c:\reports\qfe.html
```

The next block of code will process each line of the HTML file looking for a regular expression match:

```
$tmp | foreach {
  $url=$regex.matches($_)[0].Value
  if ($url) {
   $_.replace($url,"<a href=$url>$url</a>")
 }
  else {
    $_
  }
 } | Out-File c:\reports\qfe.html -encoding ASCII
```

When a match is found, which is when $url has a value, I replace the url string with a href tag that uses the $url value. If the line doesn't have a url, I still write it to the pipeline. This is necessary because all the lines from $tmp are written back the HTML file overwriting the original version.

Using Windows PowerShell and WMI is a powerful combination but sometimes the objects need a little tweaking, as I've done here.

Jeffery Hicks, MCSE, MCSA, MCT, and Microsoft PowerShell MVP, is a Scripting Guru for SAPIEN Technologies. Jeff is a 16-year IT veteran. He has co-authored and authored several books, courseware, and training videos on administrative scripting and automation. His latest book is WSH and VBScript Core: TFM (SAPIEN Press 2007). You can contact him at jhicks@sapien.com.

Exchange 2007 SP1 Returns Public Folder GUI Support

by J. Peter Bruzzese

The rumors were growing, urban legends spinning across the globe—Microsoft was dropping support for Public Folders in favor of SharePoint. The proof? No support in the Exchange Management Console for Public Folders. Further proof? Reputable tech gurus and even hints from the Exchange Team were leaning that way in posts on the Web site. Words like "de-emphasized" were being used to refer to Public Folders and their future.

Whether these rumors were at one time true is no longer the issue. If they were (and there seems to be some level of oddity in the fact that there is no GUI support in the RTM of Exchange 2007 for Public Folders seeing as how they have been around forever and they are absolutely required for pre-Outlook 2007 clients), it's no longer a concern.

The Official Word on Public Folder Support

The Microsoft Exchange Team, in an effort to calm the panic and confusion, released an update post to help explain what is really happening. It says "Microsoft will continue to support Public Folders in the next major release of Exchange Server after Exchange 2007." What this means is that there will be support for 10 years after the release of the next version of Exchange (which could be 2 or 3 years from now depending on the release schedule).

This doesn't mean Microsoft wants you to hold off on using SharePoint. The company continues to develop SharePoint and its features and hopes that you will eventually replace your Public Folder structure with a SharePoint structure. In fact, in scenarios where you're establishing a new environment with Exchange and do not have a pre-existing Public Folder structure (especially if you are working with Outlook 2007), the recommendation is to use SharePoint immediately and avoid the use of Public Folders altogether.

Setting Up Public Folder Databases

In the event you need (or want) to stick with Public Folders, however, it all begins with establishing a Public Folder database. This can be accomplished in different ways. When you perform the installation of a Mailbox Server role, you are asked if you have any legacy Outlook clients or Eudora clients in your environment. The question is an important one because if you do, you will need a Public Folder database for those clients to function. It was built-in to their requirements and there is no changing that now. Thus, by saying 'Yes' during the installation, you will automatically have a Public Folder database created for you.

If, however, you decide at some point later on—or, let's say you are in the process of a transition from Exchange 2000/2003 and need to transition over the Public Folder structure to your Exchange 2007 server and now need the database—it is easy to create one. Open the Exchange Management Console, and navigate to the Server Configuration work center. Select Mailbox. Either create a new storage group or choose an existing one within which you want to place the public folder store. Then select the option New Public Folder Database. You simply need to provide a name and the location of the database path (and decide whether you want it mounted after it is created, which is set to occur by default), and click New.

> The Standard Edition of Exchange allows for 5 storage groups and 5 databases. The Enterprise Edition allows for 50 storage groups and 50 databases. The recommendation is to place one database within one storage group.

Once the database is created (which is something you could do in RTM as well), you can go into the properties of the database (see Figure 1) and make adjustments to the

maintenance schedule, the replication intervals and limits, the storage limits and deletion settings, and age limits. You can also configure Public Folder Referral, which is configured to use AD site costs by default.

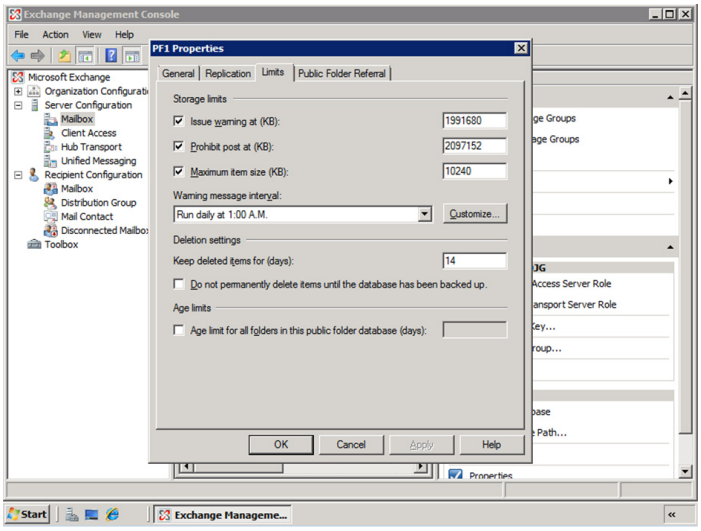

Figure 1: Public Folder database properties.

So far you might be wondering, "I thought there were no support options for Public Folders in Exchange 2007 but apparently there were." Well, to create the database, there were options in RTM. However, that is not the end all. To create the actual Public Folder structure, you had to do so through the Exchange Management Shell (EMS) or through Outlook. In addition, the configuration of Replica information and per folder storage limits was command-line driven through the EMS.

The New Public Folder Management Console

With the release of SP1, there is a new tool called the Public Folder Management Console. It's an odd location for it, I agree, but it does the job. It allows you to connect to different servers to peer into their Public Folders. You can see default and system Public Folders (see Figure 2). If you select New Public Folders (or a subfolder within), you are able to select from the actions pane New Public Folder, and create an entire structure from the GUI.

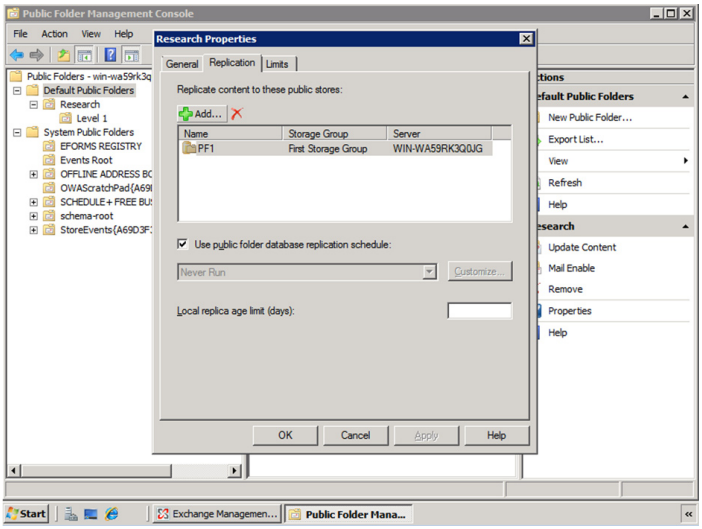

Figure 2: The Public Folder Management Console.

Once the folder is created, you can go into the Properties of the folder and work with three tabs: General, Replication, and Limits. The General tab provides information about the folder and the number of items and size. The Replication tab allows you to choose other servers on which you want to have replicas of your Public Folders. The replication schedule as well as the limits settings are taken from the database configuration settings; however, you can make changes to individual folders to allow for a stricter or looser policy on certain folders.

We Want the Truth!

Well, we might never be able to handle the truth. Were Public Folders purposely ignored in the RTM to scare people toward making that quick dive to SharePoint before SP1 fixed the problem? Who knows? But SP1 is out, and Public Folders received a much-needed new tool to help us continue to work with them. And isn't it good to know that they will be supported for many years to come!

J. Peter Bruzzese is an MCSE (NT, 2K, 2K3)/MCT, and MCITP: Enterprise Messaging Administrator. His expertise is in messaging through Exchange and Outlook. J.P.B. is the Series Instructor for Exchange 2007 for CBT Nuggets. In harmony with the joy of writing Exclusively Exchange for Realtime Publishers, he has created a free Exchange training site at www.exclusivelyexchange.com. He is co-founder of ClipTraining.com, a provider of short, educational screencasts on Exchange, Windows Server, Vista, Office 2007 and more. You can reach Peter at jpb@cliptraining.com.