

# 1

## *Introduction*

Many things in life are boring. However, there are very few things quite as boring as maintaining and configuring a fleet of NT workstations. Management of a reasonable sized network can be extremely interesting: examining network speeds, optimizing protocols, exploring new technologies, and making servers run efficiently can all be highly rewarding activities. On the other hand, is there a system administrator anywhere on the planet who doesn't sigh or groan at the prospect of cleaning up hard disks, reconfiguring workstations so they can be moved to a new location, installing driver updates, or tidying up oversized registries? The list of tedious activities can seem endless. Sure, there are plenty of tools around to help with some of these tasks: NT's built-in support for remote registry hacking, event log reading, and the like can save huge blocks of time; so too can Microsoft's SMS software. Unfortunately, however, remote access methods and "server-side push" management solutions are not always reliable, for several reasons, not least of which is that they require the workstation to be switched on when you are attempting remote access. For many mundane maintenance tasks, there is no real substitute for leaving your office and paying all your workstations a personal visit.

Well, actually there *is* a substitute. Imagine the following scenario: whenever a workstation boots up, or maybe just at a predetermined time of day, a script will run. This script might carry out some basic maintenance tasks, such as clearing temporary directories, archiving event logs, or tidying the registry. Alternatively, it might consult a server-based database and note that the workstation on which it is running was due to be moved into new offices; in preparation for the move, the script will change the machine's network identity, reconfigure IP, and change various security parameters. If any problems emerge during configuration or maintenance, the system administrator will be informed by automatic email. Finally—or perhaps initially—the script will check with a network server to make sure that it is not obsolete; if it is, it will install a replacement and kill itself without complaint.

As far as the administrator is concerned, the script is a self-regulating watchdog and servant, saving hours of time; as far as the workstation user is concerned, it doesn't even exist, as during normal operation it does absolutely nothing.

Thankfully, the previous scenario is not an excerpt from a science fiction novel or a report from sysadmin paradise. Instead, it is something that can be achieved remarkably painlessly; all it takes is a decent scripting tool and bit of imagination.

## *Automated Maintenance*

In many ways, the workstation has now come of age. From an administrator's perspective, the level of control available within Windows NT is amazing. The NTFS file system allows fine-grain setting of security privileges right down to the level of individual users and individual files; likewise, individual registry keys can be locked down on a per-user basis. Tools such as *Regedit* and the *Event Viewer* allow accessing of remote registries and event logs, respectively; the *system policy editor* forces various registry keys to adopt certain values, either globally or based on a particular user or machine.\*

While these tools are marvelous for checking the state of workstations, locking them down, and enforcing certain behavior, the very philosophy of their design leaves them with serious limitations. Essentially the tools fall into two categories:

- Those that prevent (or at least limit the potential for) user-inflicted damage or corruption. They operate by imposing restrictions on the user.
- Those that allow an administrator to access workstations remotely to carry out maintenance activities.

There are situations in which tools in both categories will be at best mildly inconvenient and at worst useless. For example, it might not necessarily be appropriate or even possible to lock down a workstation to the extent that maintenance isn't required. Since system policies and the like are designed to be preventive rather than medicinal, they provide no help curing any problems that may emerge in areas that they have not been given the responsibility for locking. Even something trivial, like an open-access temporary directory on a computer, can fill up the hard disk and cause trouble unless it is cleared regularly. Of course, a directory can be cleared remotely simply by mounting drives over the network; this would be fine for a few machines but could take hours for a few hundred. This brings us to the other problem with many conventional solutions to remote maintenance: tools such as *Regedit* are interactive. One result of this is that an administrator may not

---

\* For an in-depth discussion of such tools, see *Zero Administration for Windows*, by Craig Zacker, (O'Reilly & Associates, 1999).

have to leave her desk in order to tweak a remote registry even if the machine in question is on the other side of the world. On the other hand, tweaking the registries on even a handful of machines will take a very long time. In short, interactive tools designed to connect to single machines are not appropriate for large-scale regular maintenance.

There is no better way of carrying out many repetitive, mundane tasks than by writing a script. Such an approach is flexible and efficient; a script has to be written only once and can be deployed painlessly. Much of the first half of this book is dedicated to describing ways of building scripting solutions to a number of maintenance problems.

## *Automated Configuration*

Configuring an NT workstation might be a bore, but it usually does not have to be done very often. Writing a script to install a new driver on a single machine would take at least 10 times as long as interactively installing the driver; to choose the former option, you would need to be completely insane. However, if you have to install the driver on a few hundred machines, or even just a handful, the story changes somewhat. If you are lucky enough to have sophisticated deployment tools such as SMS, the problem *should* be trivial, although this cannot be guaranteed! What about scripting? Well, working out which registry entries need tweaking—both in relation to the new driver and dependent services—can be a hassle. However, this is easily compensated for by the fact that if you write your script correctly, you can more or less guarantee that the driver will install successfully on every single machine. This is *easily* worth the effort.

There are some instances in which rather more dramatic workstation reconfiguration is called for than mere driver installation. Imagine, for example, that a whole set of workstations needs to be moved into a new department, where they will need different machine names and different IP configuration and will even belong to a different domain. Also, because the boss is paranoid about security, they will need a new SID even though you use a domain security model for authentication. Manually carrying out this level of configuration changing is a truly time-consuming business, if for no other reason than the required umpteen reboots. A judiciously written script, however, can carry out all the changes for you without a single interactive prompt; configuration information for each workstation can be extracted from a simple text file.\*

---

\* The use of a script that completely changes a workstation identity is not restricted to moving machines between departments. In an environment in which disk-imaging techniques are used to aid rapid deployment of workstations, such a reconfiguration script can also save much manual labor.

## Scripting with Perl

Of all the tools we use in this book, by far the most important is Perl. While some of the simpler tasks we describe in the book could be written using the built-in NT batch-command interpreter, it does not take long to reach the limits of this restricted environment. We believe that Perl is a fantastic scripting language for NT workstation administration, as it combines the convenience of a scripting environment with the power of a full-fledged programming language. If you want specifics, here are half a dozen:

- Perl's regular expressions provide extremely powerful facilities for manipulating both numbers and text.
- A plethora of modules provides diverse functionality, both general and NT specific.
- Simple scripts can be written with ease and require no special environments or tools; once Perl is installed on a system, you simply grab the text editor of your choice and you're off.
- It is a multiplatform standard, so experience gained on another platform often can be applied on NT.
- It is absolutely free and well supported. Huge amounts of information and third-party modules are readily available on the Net. If you've got a problem, the online community will help you solve it.
- Command-line utilities can be executed from Perl with ease; more important, the output can be captured and manipulated. Thus the term "glue" often is associated with Perl.

Within our earshot, Perl has been called a dirty language. We take this to mean that it shuns many programming language conventions that exist in the post-C world: no strong type casting, no fixed lengths to arrays or strings, no protected namespace, and so forth. The other side of the coin is that you often hear people talk about the "right" way to write Perl, and many people take pride in writing a script the "cool" way. As far as we are concerned, we like to think of Perl as scruffy rather than dirty, and we are not at all dogmatic about exactly how a script "should" be written. On this point, we take comfort from the words of the creator of Perl, Larry Wall, and his coauthors of the Camel Book when they say "a Perl script is correct if it gets the job done before your boss fires you."

A detailed discussion of Perl is well beyond the scope of this book. However, we do describe in reasonable detail how all our scripts and scriptlets work and what every group of lines does. In addition, the Appendix, *Perl Module Functions*, gives an explanation of all the module functions that we use in the book. If you are new to Perl, and you want to do more than simply copy our scripts, then you would do

well to read *Learning Perl on Win32 Systems*, by Randal L. Schwartz, Erik Olson, and Tom Christiansen (O'Reilly & Associates, 1997); for a more thorough examination of the language, but without the Win32 slant, *Programming Perl*, by Larry Wall, Tom Christiansen, and Randal L. Schwartz (O'Reilly & Associates, 1996) is excellent.

Perl is by no means a prerequisite of using the tools and techniques detailed here. Many of them are of use in an interactive (nonscripted) environment, and all could be used with any other scripting or programming tool that provides access to the Win32 API.\* However, in order to run the example scripts that appear throughout the book, you will have to ensure that your system contains at least Version 5.02 of *ActivePerl* (see the following section); our instructions also assume that you have *.pl* files associated with Perl.†

### *Versions*

For this book, we have been using ActiveState's Perl release v5.005\_2 build 502. This was new on August 11, 1998.‡ All the scripts we have written work happily with this release and will hopefully work with all subsequent builds. At the time of writing, the current version is v5.005\_2 build 507; this was released on November 14, 1998. The release of Perl v5 for Win32 was a bit of a watershed as it marked the point at which the two most popular Perl ports for Win32 were merged. This single port greatly simplifies the process of keeping up-to-date and making sure the desirable modules are compatible.

To get hold of the latest version, download it from ActiveState's web site at <http://www.activestate.com>. Installing it is a totally painless exercise, as the download package is self-extracting and presents the ubiquitous InstallShield setup interface.

### *A Final Note*

We hope that having read this introduction you feel motivated to read on. If so, we are confident that once you have read this book and implemented some of the techniques described, you will find that maintaining and configuring an NT workstation need not be a tiresome task; in fact, it can even be quite good fun!

\* Almost all of the Perl Win32 functions that will be introduced at various points in the book are straightforward wrappers for Win32 API functions; therefore, all that is required to use them is some means of accessing the API (for example, the `#include` directive in C or `Declare Function` in Visual Basic).

† This is a more controversial point than might at first be supposed. Many Perl aficionados are no doubt cringing at our reference to *.pl* as an extension; the only "correct" extension to use would be *.plx*, for "Perl executable." However, we steadfastly stick to *.pl* because that is the default association created by ActivePerl when it installs itself.

‡ Ashley's birthday, as it happens.