

7

Changing Network Identity

Assuming that you have successfully executed at least one of the techniques discussed in the previous chapter, you will now have a foolproof way of discovering the MAC address of every one of your workstations. It is worth reiterating the significance of being able to extract this seemingly inert 32-bit word: it serves as a unique identifier, a serial number for each workstation. Armed with this number, you have a way of differentiating between machines that could be identical in every other way.

At the end of the last chapter, we listed a script that used two network utilities (*ping* and *arp*) to extract the MAC addresses from all workstations on a subnet; the script wrote out those addresses, along with IP and NetBIOS/hostname mapping to a file. We called the file *address.mac*. In this chapter, we demonstrate how a file such as *address.mac* can be used to carry out a drastic configuration change on a set of workstations: namely, to change their network identity.

There are many scenarios in which it is necessary to change the identity of a single workstation. By contrast, in most environments there are relatively few situations in which it is necessary to change the network identity of a whole set of workstations. However, if such a situation *does* arise, the ability to automate the process can be invaluable. Three such situations immediately spring to mind:

- You are deploying a set of workstations for the first time and the operating system has been installed automatically using an unattended installation script. As the same script has been run on all the workstations in question, all will share a common identity. A variation on this theme would be to have prepared the workstations with some sort of disk-imaging technique. Clearly, each machine will need its own identity before it can be released onto a network. In the case of the disk-imaged clone, it may also need its SIDs replaced (see “The SID Issue,” later in the chapter).

- You are performing a complete cleanup (including complete OS installation from scratch) of all the machines on your network. The requirement here for configuring network identities is identical to that in the preceding situation.
- A department in your organization is moving. A set of machines will need to be relocated to somewhere completely different on your network. If your naming convention is linked to topographical location rather than organizational structure, all workstations will need new names and quite possibly new IP addresses. A variation on this theme is if a set of workstations has been re-allocated to a different department in your organization. In this case, if departmental needs are similar, it may well not be either appropriate or necessary to completely reinstall the OS and effectively redeploy from scratch; all that may be required is for the hard disks to be tidied up a little and stray data removed. And, of course, each workstation will probably need a new name.

Even if you doubted the necessity at the end of the previous chapter, it should now be abundantly clear why the MAC address is such an invaluable commodity. An appropriately written script executed on every workstation can use it in conjunction with an *address.mac* file to reconfigure a machine's name, IP address, and (as will be seen later in the chapter) quite a bit more. Means of implementation, although fairly trivial, would depend on the exact scenario encountered. In the first scenario described (workstation deployment), the following stages would be required for completely automating the process of assigning network identities:

1. All workstations will have to be plugged into a network and booted into a network-capable OS. If the machines are new and contain an OS, a boot floppy with a network stack is perfectly adequate.
2. A list of MAC addresses must be compiled using one of the procedures described in the previous chapter. Machine names and IP addresses (on a non-DHCP TCP/IP network) will have to be added to this list to produce an *address.mac* file.
3. A configuration script must be written. The rest of this chapter will describe how to accomplish this.
4. The script, along with *address.mac*, will have to be installed on each workstation so that it runs when NT first boots up after installation. The most obvious way of accomplishing this would be to use a combination of automatic login and the `RunOnce` registry key (see Chapter 2, *Running a Script Without User Intervention*).

The second scenario described (complete workstation cleanup) would involve a similar procedure, but the compilation of an appropriate *address.mac* file could be totally automated using a script such as that described at the end of the previous chapter.

The third scenario (moving workstations to a new location or department) would also be similar. The means of executing a script, however, is likely to be different, as there is not likely to be an easy way of setting up an automatic logon on every machine (short of heavy manual intervention). In a situation such as this, you would be very grateful for a *stub script service* (as discussed and recommended in Chapter 3, *Remote Script Management*). If a stub is installed on all your workstations, preparing for reconfiguration will merely involve placing the configuration script (along with a newly configured *address.mac*) in the appropriate place on your server. After a single reboot of each machine, you can then be sure that the script will run when each workstation is next switched on (i.e., after it has been moved to its own location).

Retrieving Information from address.mac

The first task a reconfiguration script has to tackle is retrieving the relevant information from an *address.mac* file. This is a three-stage process:

1. Find out the MAC address of the machine on which it is running.
2. Look through *address.mac* to find a line corresponding to this MAC address.
3. Read the machine name and IP address from the line.*

We feel that the previous chapter has completely exhausted discussion on how to find out a machine's MAC address, so we will not reiterate it here. One point that is worth considering here, however, is which technique should be used to find it. If all the workstations on which configuration is to take place already have unique identities, are running on the network, and merely require a *change* in their identities, then any of the techniques we have described will work satisfactorily. However, this is certainly not the case if the workstations currently share the same identity (because they have been set up in an identical way or because they are disk-image clones). The reason for this is simple: if multiple machines all boot up with identical identifiers, then various network services will fail to start (see Chapter 5, *Controlling Services and Drivers*); any MAC-extraction technique that relies on networking may therefore fail. As successful MAC address extraction is crucial if a reconfiguration script is to work, we always grab the information from the RPC registry entry in this context (see the previous chapter).

* Although machine name and IP address are not necessarily the only parameters that a reconfiguration script will have to alter, they are the only ones we consider for the moment. In Chapter 9, *A Custom Module*, a much more sophisticated variation on the *address.mac* theme is introduced. This allows a huge number of parameters to be configured on a machine-specific basis.

Following is a complete script that will read the MAC address as stored in the RPC registry entry and will attempt to match it with an entry in an *address.mac* file. If successful, it will print out the matching computer name and IP address. Although the script is basically very simple, it highlights one very important aspect of string comparison which is certainly worthy of note: it is *crucial* to ensure that the strings you are trying to compare are genuinely comparable. While this may seem an obvious point, it is very easy to overlook. In this particular example, MAC addresses are stored in the file *address.mac* in the following format:

```
12-34-56-ab-cd-ef
```

However, the same address is stored in the RPC registry entry as a REG_BINARY value:

```
12 34 56 ab cd ef
```

When extracted as a string, this will produce the ASCII characters corresponding to the six hex values 12, 34, 56, ab, cd, and ef. This will appear as something like this:

```
-4Vx
```

(The specific characters may vary depending on the character encoding of your system.) This will clearly *never* match with a MAC address stored in *address.mac* under any circumstances. The data is stored in an incompatible format. To use the data from the registry, we need to convert it to a sequence of characters that *represent* the binary data in the same format in which it is represented in *address.mac*. We repeat, *always ensure that comparisons are made with compatible datatypes*.

That caveat dealt with, we can now look at the script itself:

```
#matchmac.pl
use Win32::Registry;
```

First, we open the relevant registry key, retrieve the `NetworkAddress` value into a variable called `$rawmac` and close the key:

```
$key = 'SOFTWARE\Description\Microsoft\Rpc\UuidTemporaryData';
Win32::Registry::RegOpenKeyEx
    (HKEY_LOCAL_MACHINE, $key, NULL, KEY_ALL_ACCESS, $RegHandle);
Win32::Registry::RegQueryValueEx
    ($RegHandle, 'NetworkAddress', NULL, $type, $rawmac);
Win32::Registry::RegCloseKey($RegHandle);
```

The next three lines deal with converting the binary data retrieved from the registry into a form comparable with a MAC address as stored in the lookup file. First, the binary data is “unpacked” into a string representation. Then a regular expression is used to append a “-” onto every pair of nonwhitespace characters (producing output such as “e0-f1-c3-04-56-34”). Finally, the unwanted trailing “-” is removed.

```
$machinemac = unpack "H12", $rawmac;
$machinemac =~ s/(\w{2})/$1-/g;
chop($machinemac);
```

Now that `$machinemac` contains a string that is directly comparable with the entries in *address.mac*, the actual lookup can take place. First, an *address.mac* file is opened for read access. Then, the script loops through the file, reading each line in turn and splitting it into the component parts (`$computername`, `$ipaddress`, and `$macaddress`). If the string `$macaddress` matches `$machinemac`, a variable `$match` is set to 1, forcing the loop to break. (The other condition that causes the loop to break is reaching the end of the file.)

```
open MACFILE, 'C:\address.mac';
$match=0;
while (($line = <MACFILE>) && (!$match))
{
    chomp($line);
    ($computername,$ipnumber,$macaddress) = split(/\s*[,;:]\s*/, $line);
    if($macaddress eq $machinemac)
    {
        $match = 1;
    }
}
```

Finally, the file is closed. If `$match` has been set (indicating that a match has been found), `$computername` and `$ipnumber` will be set to the correct values and can be used for the configuration process. Here they are simply printed to standard output. If there is no match, we print “No match found.”

```
close MACFILE;
if($match)
{
    print "Match found.\nComputername: $computername\nIPNumber: $ipnumber\n";
}
else print "No match found\n";
```

Now that we have a means of extracting information from an *address.mac* file, we can turn to the main business of this chapter, namely, using this information to configure a workstation to run successfully on a network. As the previous code detailed is somewhat verbose, we do not include it in any further scriptlets introduced later in the chapter, although it is clearly necessary for them to run. Instead, we assume that any Perl we write from now on has the requisite information available to it in the form of two scalars:

```
$ipaddress
$netbiosname
```

If you are uncomfortable with the idea of scripts referencing blatantly unassigned variables, just imagine that these values are initialized by a predefined function

whose task is to find out a machine's MAC address, compare it with a lookup file, and return the relevant data. A call to this function might look something like this:

```
($netbiosname,$ipaddress) = MCGetAddress;
```

In fact, this is not a purely hypothetical suggestion, because we have written such a function. This function is presented in Chapter 9.

Changing the NetBIOS Name

The *NetBIOS* interface is at the hub of Windows networking. Its full name, *Network Basic Input/Output System*, says it all: here is the software that gives the operating system its interface to the local area network. Both it and its more advanced relative, *NetBEUI (NetBIOS Extended User Interface)*,* have been around since the distant days of MS-DOS and Lan Manager. Just like any other network protocol that we can think of, NetBEUI requires that every computer on a network has an individual, unique identity. On a nonrouted network, it would be perfectly satisfactory to use a MAC address for this purpose. However, this 48-bit number is hardly very user friendly; an end user needs to be able to refer with ease to a computer's identity, share files, use a printer, or connect to any other network-based resource you care to mention. Hence the birth of the NetBIOS name, a text string that is used to refer to a specific computer on a network. Microsoft's networking protocols and implementations may have got more sophisticated recently, with the introduction of NetBIOS over TCP/IP, routable Windows networks, and so on, but NetBIOS names have remained.

Setting the NetBIOS Name with Control Panel

The simplest way to set the NetBIOS name for a computer is to use the Network Control Panel. Open the Control Panel, and select Network. Amusingly, the design of the visual interface to this Control Panel (as displayed in the previous chapter) itself reveals the implicit centrality of the NetBEUI protocol in Windows NT networking: the first thing you see is not a list of protocols or settings but a simple dialog tab displaying only two bits of information, NetBIOS name and Workgroup.

Clicking on Change allows either of these fields to be modified. It also gives a domain administrator the chance to create a computer account on an NT domain and get the workstation to leave its workgroup for the domain.

* To be pedantic, or maybe just accurate, we should point out that NetBEUI is not simply a more advanced version of NetBIOS. The real difference is that whereas NetBIOS is an interface, NetBEUI is a protocol that uses that interface.

Editing NetBIOS Registry Settings

As should be abundantly clear by now, GUI interfaces are hopeless for automation.* Luckily, as with most other configurable parameters in Windows, the NetBIOS name is stored in the registry and can therefore be edited with a registry editor or, of course, with a Perl script.

Two approaches can be adopted for discovering what registry entries are involved in the setting of NetBIOS names. Both approaches are equally applicable to discovering where any configurable parameter is stored, and both have been discussed at length in earlier chapters. In short, one method involves browsing the registry with Regedit or Regedt32, combining a bit of common sense with a few dollops of insider knowledge (and a dose of string-searching if you are using Regedit) to find the key you are looking for; the other method involves monitoring registry activity while you make changes using the GUI interface of the Control Panel. Whichever method you choose, the conclusion reached should be the same: a NetBIOS name is stored under the following registry entry:

```
HKLM\SYSTEM\CurrentControlSet\Control\ComputerName\ComputerName\ComputerName
```

The location of the entry is hardly surprising. Being a machine-specific setting having to do with a basic networking component of the operating system, it was almost bound to appear in `HKLM\SYSTEM`. The `CurrentControlSet` subkey (seen in Chapter 5) references the control set in current use; the `Control` subkey within this is where several parameters required for system startup are stored. Using a registry editor to browse the `ComputerName` subkey within this reveals two further subkeys, one titled `ActiveComputerName` and one simply `ComputerName`. The difference between these becomes evident if you change the NetBIOS name. Whereas the `ComputerName` subkey will contain a string value holding the new (now default) computer name, the `ActiveComputerName` subkey will contain a value holding the current computer name. After a reboot, both will hold the same data. In a script, it is necessary only to change the `ComputerName`; `ActiveComputerName` should be considered dynamic data and should not be touched.

If you try the preceding procedure, you will notice that changing the NetBIOS name using the Network Control Panel changes the IP hostname in addition to the NetBIOS name, but we will ignore this for now.

Writing a scriptlet in Perl to set the computer name registry key to a new value is trivial (presuming that we know the value to which to set it). Such a scriptlet would look very similar to those seen already in earlier chapters, the only differ-

* We should probably mention at this stage that GUI-scripting tools—such as WinBatch and Script-It—that ameliorate the problem to an extent are available.

ence being the lines of code required to assign values to `$netbiosname` and `$ipaddress`.

First, we import the `Win32::Registry` module. Then, as we don't want to clutter the script at this stage by importing our module (as mentioned earlier in the chapter) and inserting the code that retrieves the MAC address and performs pattern matching on `address.mac`, we invoke our NetBIOS/IP-address retrieval. This is used to initialize `$netbiosname` and `$ipaddress`. If you do not wish to use the module function at this stage, simply inserting the lookup script presented earlier in this chapter will have exactly the same effect.

```
use Win32::Registry;
use MaintainAndConfigure;
($netbiosname,$ipaddress) = MCGgetIdentity;
```

We then open the key, write the value, and close the key again:

```
$key='System\CurrentControlSet\Control\ComputerName\ComputerName';
Win32::Registry::RegOpenKeyEx
    (&HKEY_LOCAL_MACHINE,$key,NULL,&KEY_ALL_ACCESS,$RegHandle);

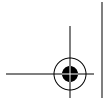
Win32::Registry::RegSetValueEx
    ($RegHandle,'ComputerName',NULL,&REG_SZ,$netbiosname);
Win32::Registry::RegCloseKey($RegHandle);
```

In the previous chapter, when we explained how to compile the file `address.mac`, we compiled it to contain MAC address, IP address, and hostname. Our discussion here of NetBIOS names has assumed that they too can be extracted from the file. The reason for this is that we use NetBIOS names and hostnames interchangeably, as it is almost invariably the case that a workstation will have the same hostname and NetBIOS name. As we have mentioned, it is assumed to such an extent that changing the NetBIOS name through the Network Control Panel also changes the IP hostname.

Setting Up IP Networking

IP is probably the most widely used routable networking protocol in the world. It is also probably the best documented. Countless higher-level protocols have been designed to work over IP, most notably TCP (for one-to-one guaranteed stream connections between machines) and UDP (for datagram broadcasts). A discussion of IP, let alone TCP/IP or the simpler UDP/IP, is well beyond the scope of this book; for an excellent and thorough introduction, read *Windows NT TCP/IP Network Administration*, by Craig Hunt and Robert Bruce Thompson, (O'Reilly & Associates, 1998), as noted in the previous chapter.

As far as we are concerned here, the significance of IP is that machines running it require an IP address. Further, this IP address has to be unique on a network; if



the machine in question is connected to the Internet, then the address must be unique on the Internet. Unlike NetBIOS names, IP addresses are used for communication beyond the bounds of a local subnet; therefore, an incorrect IP address (in conjunction with a badly configured router) can cause general havoc. Clearly, it is only right that the IP address should be one of the fields in *address.mac*!

Using Control Panel

The simplest way to configure an IP address, just like with NetBIOS name, is by using the Network Control Panel. This time, switch to the Protocols tab, select TCP/IP protocol and click Properties. You will be presented with a dialog box (see Figure 7-1) containing tabs for IP Address, DNS, WINS Address, and Routing. In virtually every conceivable situation, you will want the DNS, WINS, and routing information to be the same for every machine you are configuring,* so we are primarily interested only in the IP Address dialog, as the address information will have to be set on a workstation-by-workstation basis.

Figure 7-1. TCP/IP Properties dialog box

* This assumption may be violated, of course, on an exceptionally large network.

For each adapter in the machine (most workstations will contain only one), this sheet provides fields to specify the IP address itself, a subnet mask, and a default gateway. Changing the information is a simple matter of typing in the replacement information and clicking Apply.

There is one bit of information in the DNS sheet that is machine specific, and that is the hostname, the text-based alias to an IP address. DNS was briefly discussed in the previous chapter, so the discussion will not be repeated here. Suffice it to say that a fully qualified hostname (the hostname and the name of the domain in which the machine resides) should be unique for every workstation and thus needs to be configured.

We have already noted that it is almost always true that a hostname corresponds to the NetBIOS name and that if you change the NetBIOS name via the Control Panel, the hostname is also changed. It is worth noting, however, that this is not an absolute requirement. It is perfectly possible to change the hostname independently, although the Control Panel chooses to warn you if you do that:

The new Internet host name is different than the computer name currently in use. Your computer will now be known by one name on your local area network, and by another name on the Internet. If the computer name is ever changed it will overwrite this new host name and the changes will be lost.

If you want to write a script that configures these two parameters separately, an extra field will be needed in *address.mac*, and the code that extracts the information will have to be adapted accordingly.

It is also worth noting that the hostname parameter here is nothing to get hysterical about: whereas the NetBIOS name is vital for NetBEUI to work, the hostname parameter stored on the local machine is barely used by the OS at all. After all, the point in DNS resolution is that other machines can find your machine by asking a server what your IP address is; your machine already knows what its own IP address is, so is little bothered by its hostname. In this light it could be argued that Microsoft's warning message is a little misleading.

Changing IP Registry Settings

Changing all the entries in the IP Address dialog and setting the hostname under DNS while running registry monitoring software will reveal that a huge number of entries have been written. This is because all of the IP-related network settings are rewritten even if they have not been altered. Although at first sight this seems a little overwhelming, it does not take long to isolate the entries of interest. All of them are string values (REG_SZ). On our machines they are:

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Hostname  
HKLM\SYSTEM\CurrentControlSet\Services\PCNIN4M1\Parameters\Tcpip\IPAddress
```

```
HKLM\SYSTEM\CurrentControlSet\Services\PCNTN4M1\Parameters\Tcpip\SubnetMask
HKLM\SYSTEM\CurrentControlSet\Services\PCNTN4M1\Parameters\Tcpip\DefaultGateway
```

These keys, as is clear from the names, store the Hostname, IP Address, Subnet Mask, and Default Gateway values for the Current Control Set. Given the nature of the values, it is unsurprising that they all reside within the `HKLM\SYSTEM\CurrentControlSet\Services` subkey or that within the specific service subkey they are stored within a further `Parameters` subkey; after all, TCP/IP is a service, and all these values are parameters of the service. What does require explanation is why the hostname is a parameter of the TCP/IP service, whereas the others are part of the PCNTN4M1 service. This is especially worthy of note, given that PCNTN4M1 is specific to the machine on which we happen to be working at the moment (the keyname will probably be different on your machines).

In fact, there is nothing particularly weird here. A workstation can have only one hostname* (it can have several DNS entries, but this is not the business of the workstation itself); it therefore makes sense to store the hostname under the parameters section of the TCP/IP services subkey. On the other hand, a workstation may have more than one network card, and each card can have more than one IP address (and associated settings). Therefore, it would make no sense for IP Address, Subnet Mask, and Default Gateway to be stored under the TCP/IP parameters key. Instead, each set of IP-related information is stored in its own subkey. In the case of our machine, "PCNTN4M" is the name of the NIC driver, and the key "PCNTN4M1" is used to store the configuration information corresponding to the first (and only) set of parameters attached to this card. The rest is self-explanatory.

Provided that all the machines you are configuring reside on the same IP subnet, the only settings that will have to be changed by a script on a workstation-by-workstation basis are the hostname and IP address. The default gateway should remain identical for them all; further, unless you have an extraordinary network setup, the subnet mask will also be the same for all machines.

The format of a Perl script that makes changes to the registry should be rather familiar by now. A script that sets up hostname and IP address follows. As in the case of the NetBIOS-name-changing scriptlet, we will assume that the function `MCGGetIdentity` is at our disposal. Given that registry functions in Perl should now be familiar, we do not provide any comments here:

```
use Win32::Registry;
use MaintainAndConfigure;
($netbiosname,$ipaddress) = MCGGetIdentity;
$key='System\CurrentControlSet\Control\ComputerName\ComputerName';
Win32::Registry::RegOpenKeyEx
```

* This is not actually true on a multihomed system. However, Microsoft seems to have ignored this, so so do we.

```
(&HKEY_LOCAL_MACHINE, $key, NULL, &KEY_ALL_ACCESS, $RegHandle);  
Win32::Registry::RegSetValueEx  
($RegHandle, 'ComputerName', NULL, &REG_SZ, $netbiosname);  
Win32::Registry::RegCloseKey($RegHandle);
```

Unless the machines that need reconfiguring with new identities all reside on the same IP subnet, the chances are that one further bit of information will need to be changed on a workstation-by-workstation basis, namely, the IP address of the default gateway. A gateway is a door to the subnet: every IP packet destined for a machine that is not on the local subnet (read, “Cannot be resolved by *arp*”—see Chapter 6, *Machine-Specific Scripting*) is sent off to the default gateway, a router that should know what to do with the packet. Finding out which registry setting holds the gateway IP address is simple enough—it is listed a few paragraphs earlier—but what should we set it to?

The answer to that question depends to a large extent on the size and the IP-address-allocation convention of your network. If you are unlucky, setting the default gateway will require yet another field in *address.mac* and another corresponding change in the script that reads this file so that it can get at default gateway information for each machine. This would presumably lead to the initialization of another Perl scalar called something like `$default_gateway`. However, this is normally unnecessary for two reasons:

- A default gateway is likely to be on the same subnet as you. The first three bytes are therefore going to be the same as the first three octets of your IP address.*
- If you run your own DNS/IP registration, or it is run by sane people, it is likely that all routers on your network will share a common fourth octet.

Using these two facts, it is possible to write a Perl script that deduces the IP address of a default gateway on the fly.

For example, if machines on one subnet all have the IP address prefix 123.234.345, machines on another subnet all have the prefix 123.234.543, and your IP-allocation convention dictates that routers’ IP addresses have the suffix .90, then you can deduce the following about default gateway IP addresses:

- The default gateway on the 123.234.145 subnet has the IP address 123.234.145.90.
- The default gateway on the 123.234.541 subnet has the IP address 123.234.541.90.

In the following script fragment, we extend the writing IPAddress and hostname scriptlet so that it also writes the registry value for default gateway, based on the assumption that the IP address suffix will always be .90 and that we are always

* Our reference to the first three bytes here assumes a Class C network; on a Class B network, the default gateway may share only two bytes with your machine.

dealing with Class C subnets. In order for this fragment to work, it will have to be appended to the previous script (as it has no use statements). Here, we set `$ipaddress` for the sake of clarity in the example. The script additions are:*

```
$ipaddress='123.234.211.111';  
$routersuffix='90';
```

Open the registry key in which the default gateway is stored:

```
$key='SYSTEM\CurrentControlSet\Services\PCNIN4M1\Parameters\Tcpip'  
Win32::Registry::RegOpenKeyEx  
(&HKEY_LOCAL_MACHINE,$key,NULL,&KEY_ALL_ACCESS,$RegHandle);
```

The regular expression that follows matches “three sets of one or more digits and a .” that are followed by “one or more digits.” It stores “three sets of one of more digits”—i.e., the subnet prefix—as `$1`.

```
$ipaddress=~ /((\d+\.){3})\d+/;
```

We set `$gateway` to be a concatenation of `$1` and the router suffix:

```
$gateway = "$1$routersuffix";
```

Finally, we write the registry value and close:

```
Win32::Registry::RegSetValueEx  
($RegHandle, 'DefaultGateway', NULL, &REG_SZ, $gateway);  
Win32::Registry::RegCloseKey($RegHandle);
```

Cosmetics

Configuring network parameters such as NetBIOS name and IP address may be the only essential steps required to give an NT workstation a new network identity. However, there are a few finishing touches (and one in particular) that are of no practical relevance whatsoever but that can make the real difference between mediocrity and perfection on a reconfigured workstation. What are these “touches” of which we speak? They can be anything, really, from machine-specific desktop icons to customized login dialogs. An explanation of how to make either of these cosmetic changes is beyond the scope of this book (although with a bit of ingenuity and some help from Chapter 9, nothing should be impossible). Here, we concentrate on only one cosmetic feature of the operating system, which even has practical value, namely, the My Computer icon.

This icon (system policies permitting) sits on the desktop of every NT workstation regardless of who is logged in. It contains icons that provide links to drives (both local and network), printers, the Control Panel, and even dial-up networking if this

* It is probably not very good practice to hardcode potentially changeable information in this way, but we do it in this example to make it absolutely clear what `$ipaddress` and `$routersuffix` are.

is installed. However useful this shortcut may be, we have always been very irritated by the icon, not because of its design but because of the text. If you are sitting at your own computer, you do not need to be told so; if you are not, the text “My Computer” is actually misinformation. In other words, the icon is at best uninformative and at worst a propagator of misinformation. All right, maybe we are making a bit much of this, but how much better it would be if the text “My Computer” were to be replaced with a string giving the computer’s real name.

Given that our scripts already require us to know the NetBIOS name/hostname, it should be a trivial matter to assign this string to the My Computer icon. After all, the chances that the text is stored in the registry somewhere are fairly high. Using the tried and tested entry-finding procedure (switch on monitoring, change the text beneath the icon—click on it and type something new—and see what comes up), NTRegmon (or equivalent) will present you with the following key:

```
HKEY_CLASSES_ROOT\CLSID\{20D04FE0-3AEA-1069-A2D8-08002B30309D}
```

It is the default value that is changed; this is a string (REG_SZ) value.

This rather bizarrely named registry entry could just be added to the Perl configuration script; then all workstations would reconfigure themselves with a new text string beneath the My Computer icon. (We assume that at this stage in the chapter an example of a script to do this would be superfluous to requirements.) However, we feel that the key requires just a little further explanation. After all, we have already come across some slightly unintuitive registry entries, but this string of seemingly random numbers and letters is something else. It is especially weird considering one might have imagined that one was looking for something along the lines of:

```
HKLM\SOFTWARE\Microsoft\Explorer\MyComputerIcon\Message
```

The truth of the matter, of course, is that there is nothing the least bit surprising about this keyname, and if you look in the registry under the subkey `HKEY_CLASSES_ROOT\CLSID`, you will see a veritable cornucopia of such keys. What are they? Well, it’s obvious isn’t it? They are CLSIDs.

The *CLSID* (*Class ID*) was born with OLE, a technology that has now evolved into *COM* (*Common Object Model*). CLSID is a specific kind of *GUID* (*globally unique identifier*), a 128-bit number generated randomly. The point in a GUID is that the chance that the same one will be generated twice is very remote; therefore, if you need a persistent, unique identifier, a GUID is for you. In the case of CLSIDs, the identifier is used to represent an OLE class object; a registry key such as the one described earlier stores information about the object.

Why is the text of the My Computer icon stored as a parameter of a CLSID? Because it *is* an OLE object, specifically, it is part of Explorer.

The SID Issue

So far in this chapter, we have discussed parameters that need to be set uniquely for every workstation in order to allow networking to work successfully; we have also discussed the My Computer icon. In many situations, no more steps will need to be undertaken in order to complete the reconfiguration process, other than to restore the stability of each workstation (for example, disabling automatic login—see Chapter 8, *Script Safety and Security*) and perform a reboot. There are some situations, however, in which further work will be required. Specifically, it is possible that it will be necessary—or at least desirable—to reset the workstation SIDs.

What Is a SID?

A SID is a 96-bit identifier, unique for every workstation and user; it forms the basis of NT security.* Whenever you attempt to access anything on a workstation or use a network resource, the “access token” associated with your account is compared with the “access control list” of the resource; the access token contains your user SID, group IDs, and user rights policies.

A workstation SID is created during the NT setup process; the setup program simply produces an identifier, which, due to the comparatively large number of bits, has a good statistical chance of being unique. When a user account is added to the workstation, this is created by appending a *RID* (*relative identifier*) to the workstation SID; the RID is simply a number that increments by one for every user. For example, if the workstation account SID is:

S-1-5-21-1070621424-2124630364-1947940980

then the first normal user account (not the special administrator and guest accounts) that is created would have a SID of:

S-1-5-21-1070621424-2124630364-1947940980-1000

A second user account would then have a SID of:

S-1-5-21-1070621424-2124630364-1947940980-1001

When Would You Want to Change a SID?

The most likely scenario in which you might want to change a SID is when a set of machines have been deployed with some sort of disk-image cloning system. This would create a situation in which multiple workstations shared a common SID, which can be problematic as anyone with access to resources on one machine would suddenly have access to all. Even if you delete all user accounts

* The security model is liable to change quite substantially under Kerberos and Windows 2000.

and then create new ones, the new user SIDs will be the same on every machine. The reason for this is simple: all workstations will have the same computer SID because this is created when the OS is installed (and clearly not recreated during a disk-imaging process); as RID generation is not machine specific (normal user RIDs start at 1000 and are incremented by 1 each time a new user SID is needed), users on different cloned workstations will have the same SID. It is this very fact that led to a *Microsoft TechNet* article (Q162001) entitled “Do Not Disk Duplicate Installed Versions of Windows NT.”

While it is certainly true that workstation security goes out the window if different users on different machines share common SIDs, the problem disappears altogether in a domain environment. If user accounts reside on a domain controller instead of on a workstation, then they will be created from the Primary Domain Controller (PDC)’s SID, and the workstation becomes irrelevant. To be sure, every machine will have the same local administrator and guest SIDs, but this really should not matter a jot. After all, if you are the administrator of a whole set of machines that share a common SID, then you *want* to have administrative rights on all the machines, so having the same SID on each one is perfectly sensible and logical. As far as the local guest account on each workstation is concerned, if you are worried enough about security even to know what a SID is, then you will have disabled this account long ago!







The *Microsoft TechNet* article states that “simply copying the contents of one hard disk to another would give each computer the same SID, making security impossible to maintain.” Well, they are right about the first half, but in a domain environment, security is perfectly easy to maintain and not even marginally compromised by workstation cloning; just don’t use local workstation accounts.

For fear of being branded lax on security, however, we will stress that *if you are not running a domain environment, and/or your workstations contain active local accounts, then you should always change SIDs on a workstation that has been deployed using a disk-imaging technique.*

There are occasionally other situations in which changing a SID might be considered a good idea, such as a gross security breach on your network, a paranoid employer, and so forth.

Changing a SID

The bad news is that changing a SID is far from being a trivial issue: SIDs are littered all over the registry, and they are associated with the access control lists for all files and objects. The good news is that there are programs available that will whirl through your computer, changing every occurrence of the computer SID with a brand new one. Here we discuss only one of these programs, *NewSID.exe*.

Although it is not the only one available,* it is reliable and a good representative of its type.

Using NewSID.exe

NewSID is a program written by Mark Russinovich and Bryce Cogswell of NT Internals fame. As is their wont, their program is:

- Free
- Published with full source code
- Accompanied by a manual that really tells you how the thing works

It can be downloaded from Systems Internals at <http://www.sysinternals.com>.

Running *NewSID* is incredibly easy. To start it in interactive mode, simply type `newsid` at the command prompt. A small dialog box is displayed, with four buttons labeled as follows:

- Apply New SID
- Synchronize SID
- Computer Name
- Cancel

The Computer Name option duplicates functionality that is built in to the standard Network Control Panel, namely, it allows you to replace the NetBIOS name of a workstation.† Apply New SID starts the process of creating a new SID, and replacing every instance of the old one. This includes scouring the Security and SAM (Security Account Manager) registry hives, the registry key access control lists, and the access control lists associated with all files in an NTFS filing system. Synchronize SID provides an alternative method of generating the new SID to be applied: rather than creating one randomly, synchronization allows a SID to be taken from another machine. This would be useful if you were configuring a backup domain controller, as such a beast must have its SID provided by the Primary Controller for the domain. The cancel option is self-explanatory. As we have mentioned repeatedly throughout this book, GUIs are not very good in a scripting environment; *NewSID* can be run noninteractively. Typing `newsid /a Dolly` at the command prompt would replace the SID, change the NetBIOS name to *Dolly*, and—assuming these steps were successful—reboot the machine.

* There is now even an offering from Microsoft itself.

† Although this functionality is unnecessary (as name changing is easy to implement), it is a nice feature to provide; after all, if you are going to the trouble of changing a SID, you are likely to also want to change the machine name.

A word of warning

The documentation that comes with *NewSID* includes an ominous disclaimer, warning of the dangers of running “any software that changes file and Registry settings.” This is certainly true, particularly for software such as this, which is changing values that are virtually unreadable to a human. While we have found *NewSID* to be a very reliable program, we would not recommend using any such product unless absolutely necessary. There is one simple reason for this: SID changers take quite a substantial time to run (several minutes); we dread to think what might happen if, say, there was a power cut while metamorphosis was in midflow. We suspect that the chances of being able to salvage your workstation without a total OS reinstall are virtually zero!

Access to an NT Domain

If your network relies on the NT domain model for security, there is one final problem that may emerge in *any* situation involving reidentification of workstations, whether reidentification is taking place during workstation deployment or if the extent of reconfiguration is limited to a mere name change. The problem will occur whether configuration is being carried out manually or automatically: every workstation is likely to lose access to its NT domain. When a workstation first joins a domain, a *secure tunnel* is created between the workstation and a Primary Domain Controller; this is an encrypted connection between the two machines over which the NetLogon service (the service responsible for network logons) can communicate privileged information safely. In order to use this secure tunnel, a workstation must supply a password to the PDC, a password that is generated when the workstation first joins the domain and that is stored in an encrypted part of the registry (the Local Security Authority, or LSA). This security information is specific to every workstation, identifiable by *name*; therefore, changing a computer name can render it unable to log on to a domain.

If a renaming procedure is carried out manually, none of this is an issue. The facility for joining a domain (thereby recreating a secure tunnel) presents itself on the very same tab of the Network Control Panel as does the facility for changing the computer name in the first place. When changing a name, it is a trivial matter for a domain administrator to recreate a domain account as part of the same procedure. Unfortunately, life is not so simple for a script: as creation of a domain account and the joining of a domain using that account involves interaction between a workstation and a PDC, no amount of registry tweaking or local configuration will allow this to be achieved. Short of programming the API directly, the only way to automate the process of joining a domain is to use a command-line utility.

When looking for a solution to a problem such as this, the first port of call is normally the extremely useful *net.exe*. As has been noted previously, this program contains all sorts of features for controlling Microsoft networking. One of the invocations of the commands, `net computer \\computername /add`, looks at first glance as though it is exactly what is required. Unfortunately not. The first limitation is that this command can be run only from a server. The second is that even if a server is at hand, *creating* an account for the computer is not the end of the story; getting a workstation actually to *join* a domain (i.e., to use the account that has been created) is a very different matter.

Thankfully, there is a solution. Supplement 2 of the *NT Server Resource Kit* comes with a utility called *Netdom.exe*, which can carry out exactly the task required. Although *Netdom* seems to be a good program, and it has a very sensibly designed command-line frontend, the requirement to use it does not please us. The reason is that it seems mildly perverse that something so central to Windows NT's network security model as the ability to join a domain cannot be scripted without the use of a utility that has to be purchased as a supplement to a server resource kit. Having said this, the utility does exist, so we will now turn to describing how it can be put to use.

Using Netdom.exe

A variety of tasks can be carried out with *Netdom.exe*, all concerned with either querying or stating the relationship of a workstation to a domain. Here we are concerned with only one of its incantations, the one that allows a workstation to join a domain. The syntax for this incantation is as follows:

```
NETDOM Only /Domain:Domain /user:user /password:password MEMBER mycomputer  
/JOINDOMAIN
```

The first parameters here allow specification of a domain and a set of credentials of a user running the command. (In this context, the user specified must have the rights to add computers to the domain.) The declaration *member* specifies that the arguments that will follow relate to the part of the program that deals with altering the relationship between domain members and their controllers; this syntax is similar in style to that of *net.exe*. The final two arguments specify the name of the computer and that the action that should be performed is joining a domain. Invoking the preceding command on a workstation will carry out the following actions:

- Create a computer account for the workstation on the PDC, if one does not exist already.
- Set a secure tunnel between the workstation and the PDC.

- If the workstation is currently configured as a member of a workgroup (i.e., the login dialog does not present an option for specifying domain), it will set it to domain mode.

Incorporating a call to *Netdom* in a configuration script will ensure that after a reboot each renamed workstation will be able to connect nonproblematically to its domain controller.



In Chapter 2, we discussed the security implications of credential information (i.e., username and password) for users with administrative privileges being stored in scripts on workstations. Everything we noted there applies even more so when applied to scripting *Netdom*, as such a script will contain credentials of a user with certain domain-administrative privileges. Ensure that the script cannot be read by nonadministrative users and preferably that it deletes itself after application. Finally, do *not* use a domain admin account, just in case someone does find your script. The only domain-level privilege that *Netdom* needs is the ability to add a computer to a domain; create a special account with only this privilege and use this in your scripts. (This issue is discussed further in the next chapter).

Summary

We started this chapter by explaining how to retrieve workstation identity from a lookup file based on a MAC address. We then introduced the various parameters that need to be configured in order to change the identity of an NT workstation and explained how to change them. This included a discussion of NetBIOS names, TCP/IP parameters, and a particularly pernicious Explorer icon. This was followed with a brief look at SIDs, explaining why, when, and how to change them. Finally, we discussed the issues associated with changing computer identities in a domain environment.

The chapter has shown how to automate a procedure that would take substantial administrative time to accomplish if undertaken manually. In so doing, it has brought together many of the techniques introduced in previous chapters and shown how they can be used to carry out drastic configuration tasks with relative ease. In the process of running, the types of scripts we have discussed in both this and previous chapters may carry out all sorts of maintenance and configuration tasks, but they can also leave workstations in unstable states. For example, automatic log ins may be set, or networking may be disabled. In the next chapter, we discuss how to ensure that configuration scripts run reliably as anticipated and remove themselves gracefully from a workstation, so that when they have finished their tasks, they leave no unhappy side effects.