Buried treasure

Roger Gann digs deep into the Redmond files and reveals a treasure trove of undocumented DOS dodges to make your life easier.

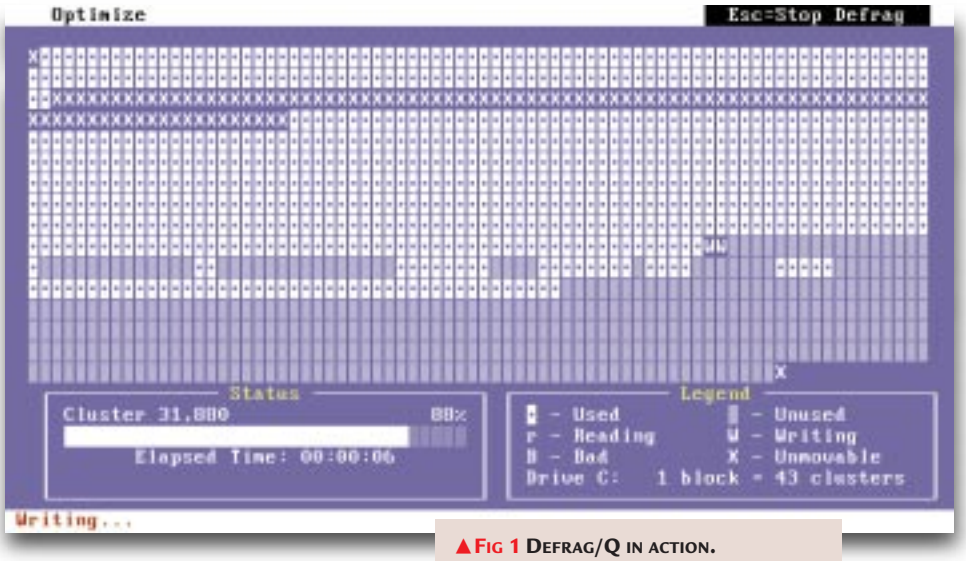
In last month's column I revealed some fascinating undocumented DOS titbits and promised more for this month. So, true to my word, here they are, pulled from the X-Files in deepest Redmond! Once again, these vary in importance from trivial to intriguing. Note that most, but not all, will work with Windows 9x.

1 DEFRAG /Q. While Windows 9x isn't so fussy, Windows 3.x requires contiguous free space on a hard-disk drive in order to create a permanent swap file. You create this contiguous space by running a file defragmenter (such as DEFRAG, included with MS-DOS 6.2) which moves all the files on a disk, leaving a single, unbroken, vacant space on the drive.

☛ **You can create** a permanent swap file in this space by opening the Windows 3.1 Control Panel's 386 Enhanced dialog box, clicking the Virtual Memory button and then clicking the Change button. Normally the DOS utility, DEFRAG, does two things: it rearranges each file's clusters into consecutive order, and it makes all files contiguous, leaving one large open space on the drive. But this process can be slow, unless you use an undocumented switch which speeds up DEFRAG.

☛ **Run the command** DEFRAG /Q (for Quick) at a plain DOS prompt with Windows not running [Fig 1]. DEFRAG now doesn't bother to move each file's clusters into consecutive order. Instead, it simply moves all of the disk clusters so that they reside at the beginning of the drive, leaving one contiguous space on the disk — perfect for creating a permanent swap file!

☛ **Having made** your swap file (a process that takes only a few seconds in Windows) you may want to defragment all the files completely because this gives the user faster disk access. In that case, start a full defragment with the command DEFRAG /F after exiting Windows. Not Windows 9x!



▲ **Fig 1 Defrag/Q in action. Cosmetically indistinguishable from normal modes, the /Q option merely 'compacts' all the clusters, thus saving time**

2 Break the 640Kb barrier. Sometimes 600Kb of free memory isn't enough when DEFRAG is run on a badly fragmented large disk. If you're running DEFRAG on a 386 or 486, there's an undocumented workaround. If you're using a VGA graphics card and monitor, load EMM386.EXE with the command

```
DEVICE=C:\DOS\EMM386.EXE
NOEMS I=A000-B7FF NOHI
```

in CONFIG.SYS. For a system with monochrome video, use the command

```
DEVICE=C:\DOS\EMM386.EXE
NOEMS I=A000-AFFF NOHI
```

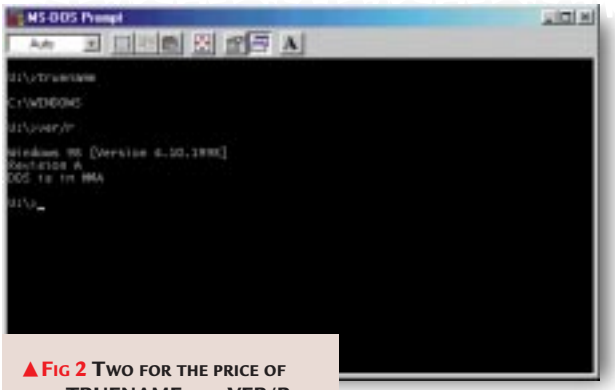
Configured this way, EMM386.EXE converts a portion of the area normally reserved for the video buffer to usable RAM and links it with conventional memory, resulting in maximum executable program sizes approaching (and sometimes even exceeding) 700Kb. This should give DEFRAG plenty of room to run. If your system uses an extended BIOS, the extra RAM will not be linked with conventional memory. Instead, you'll gain an extra 96Kb of UMB (upper memory block) RAM, or 64Kb on monochrome systems.

Note that most, but not all, will work with Windows 9.x

☛ **You might think** that this is a great way to get more memory for RAM-hungry games, but sadly it isn't. Since this workaround uses memory normally reserved for your graphics adapter, you can't run any graphics-based programs while your system is set up in this way. Since DOS 6.0's DEFRAG program is a text-only application, you won't run into problems.

3 INSTALLHIGH. You are probably aware of the INSTALL command, which loads TSRs from CONFIG.SYS, rather than AUTOEXEC.BAT. But that's only half the story. DOS 6.0 also supports a similar but undocumented command, INSTALLHIGH. This loads TSRs into upper memory from CONFIG.SYS in much the same way that LOADHIGH loads them into upper memory from AUTOEXEC.BAT.

☛ **You could argue** that it's better to load TSRs from CONFIG.SYS simply because it accepts the question mark parameter (?) and allows for the F5 and F8 keys, which let you skip CONFIG.SYS or pass over it one line at a time. Adding the question mark to a statement that loads a driver or a TSR makes DOS prompt you before executing the program. This is useful when you don't want to waste memory loading a program you don't need at every session.



▲ FIG 2 TWO FOR THE PRICE OF ONE: TRUENAME AND VER/R IN ACTION, UNDER WINDOWS 98

For example, enter this command in CONFIG.SYS to generate a YES/NO query before DOS attempts to load the program:

```
INSTALLHIGH?=C:\MOUSE\MOUSE.COM
```

Press Y to load the program or N to skip it.

☛ **To load TSRs** and device drivers high, you need to have EMM386.EXE loaded in CONFIG.SYS with INSTALLHIGH commands after it. But there's a small caveat: INSTALLHIGH doesn't support the LOADHIGH /L switch available in AUTOEXEC.BAT, which allows you to specify the memory region that your program loads. A TSR loaded with INSTALLHIGH is always placed in the largest free upper memory block.

4 SWITCHES = /W. This is one for users of older versions of DOS. It is not relevant to Windows 9x. MS-DOS 5.0's SWITCHES command supports an undocumented switch that enables Windows 3.0 users to move the file WINA20.386 out of the root directory. Usually, this virtual device driver can work only if it's located in the root directory. However, add a SWITCHES=/W statement to CONFIG.SYS and a DEVICE command specifying the new file location to the [386Enh] section of SYSTEM.INI, and you can move WINA20.386 wherever you like. So, if you moved the file to your \WINDOWS\SYSTEM directory, the line would look like this:

```
[386Enh]
DEVICE=C:\DOS\WINDOWS\SYSTEM\WINA20.386
```

Note that WINA20.386 is not required for Windows 3.1 or 3.11.

5 Long paths. One of the long-standing shortcomings of DOS is the parsimonious 127-character limit it places on commands. In fact, it's a bit less than this; the PATH statement actually limits the length of the PATH environment variable to 122 characters (plus five for PATH=). DOS 6.0 finally breaks through this limit by allowing PATH statements to be created in CONFIG.SYS with a SET command instead. To set up a long path statement, open AUTOEXEC.BAT in EDIT (DOS' text editor).

☛ **Move down** to the PATH statement, press Home, press Shift+End to select the entire PATH line, and press Shift+Del to cut the statement.

☛ **Without exiting EDIT**, press Alt+F, O to open a file, type in CONFIG.SYS and press Enter. Press Y when prompted to save the file and CONFIG.SYS appears in the editing screen.

☛ **Press Ctrl+End** to get to the bottom of your document, and press Shift+Ins to paste the old path into CONFIG.SYS.

☛ **Press Home** to return to the beginning of the new path line, and insert SET and a space before the path statement. For example, this statement adds DOS, WINDOWS, and UTILS to the path environment:

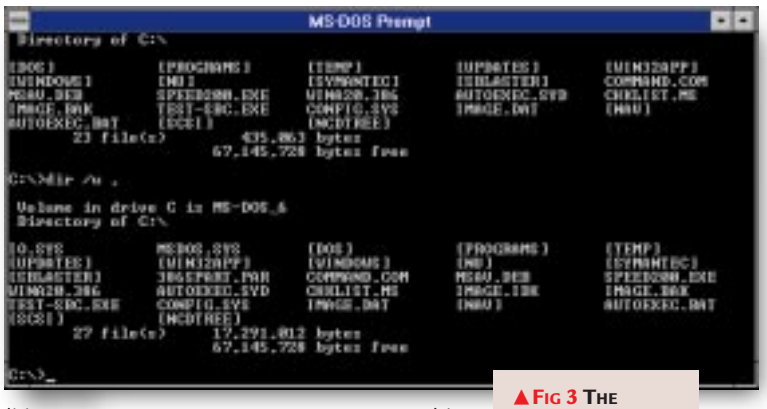
```
SET PATH=C:\DOS;C:\WINDOWS;C:\UTILS
```

☛ **Press End** to reach the end of the statement and add more directories separated with semicolons without worrying about the line's length.

☛ **Press Alt+F, S** to save the file, and Alt+F, X to exit EDIT. Reboot the computer in order to make your changes take effect.

You don't get owt for nowt, though, and this method has a couple of drawbacks. Firstly, DOS won't show any path directory entries past the 127-character limit when you enter PATH at a DOS prompt. Secondly, you can't alter paths on-the-fly, as you could when the PATH was specified in AUTOEXEC.BAT: you cannot, for instance, manipulate it in a batch file using the %PATH% variable either.

6 TRUENAME. This is another classic DOS secret which first surfaced with MS-DOS 4.0. Pass TRUENAME a filename, and it returns a fully "qualified" filename. That is, one that includes a drive letter and a complete path to the file from the root directory. Its most important use is probably on networks where drive mappings and use of the SUBST command hide the real path from the user. TRUENAME can "see through" aliases created by commands such as SUBST and other network utilities.



▲ FIG 3 THE ELUSIVE COMMA MAKES ALL THE DIFFERENCE, REVEALING HIDDEN OR SYSTEM FILES

7 VER/R. This is an odd one [Fig 2]. Enter VER with the undocumented /R switch and, in addition to reporting the version and revision of DOS on your PC, it will also tell you whether DOS is loaded into high memory area (HMA). This information is useful if you have to reserve HMA space for, say, NetWare shells.

8 DIR. Another trivial oddity. By adding a humble comma [Fig 3] to the DIR command, it will list all System/Hidden files, rather like DIR/OS but saving you a few keystrokes. This one doesn't work under Windows 9x.

PCW CONTACTS

Roger Gann *can be contacted via the PCW editorial office (address, p10) or email* **16bit@pcw.co.uk**