



APC's monthly pot-pourri of hardware and software productivity tips. APC will pay between \$50 and \$200 for each tip published. Send your tip on disk and in printed form to TJ's Workshop, APC, 47 Glenhantly Road, Elwood 3184.

FDISK rescues a hard disk drive

I recently installed a 43Mbyte hard disk in an IBM PC-XT. The hard disk came from an old minicomputer that was to be scrapped. For the cost of a disk controller I was able to upgrade an XT with a badly needed hard disk.

The low-level format ran without problems. I used FDISK to set a primary partition of 32Mbytes and an extended partition for the remainder. Everything looked great until the FORMAT command bombed. After a couple of attempts, I realised that FORMAT would always fail at the same place on the hard disk, which rendered 15Mbytes unusable. My first thought was to use FDISK to partition around the affected area. FDISK let me set the primary partition up to the offending cylinder, but insisted on using the bad cylinder as the starting value for the extended partition. After a couple of cups of coffee and a little head scratching, I found the solution.

I set the primary partition to cover all of the disk area up to the affected area, as before. But this time, I created a small extended partition to cover the bad cylinder. Then I set up an additional extended partition using the remainder of the disk. FDISK labelled the three partitions as C, D and E. Next, I used FDISK to delete the D partition (the problem area). It changed the last partition from E to D, thus effectively blocking out the offending area. The hard disk has been humming along ever since.

J McCoy

DOS can map out a partition's bad sectors in the file allocation table (FAT). However, if a bad sector is located right in the FAT or in the partition's root directory, FORMAT can't handle it. This solution is clever: it creates a partition that involves only the bad sectors and then removes all access to that partition — NR.

Faster formatting

The FORMAT command in PC-DOS 3.3

has an undocumented switch that makes formatting disks faster. The /H switch makes FORMAT skip the 'Insert disk and press any key' message:

```
FORMAT A: /H
```

This immediately causes the diskette in drive A: to be formatted. After formatting, DOS will ask if you want to format another. Insert another disk, type Y, hit Enter and FORMAT will immediately format that disk.

I discovered this undocumented switch when using the BACKUP command's /F switch. If BACKUP encounters an unreadable disk and you specified the /F switch, it will start formatting the disk immediately. I knew BACKUP was EXECuting FORMAT, so there had to be a switch on the FORMAT command that caused an immediate format. Dumping BACKUP.COM led me to the /H switch.

I don't know if this works under MS-DOS or any other OEM versions of DOS, but it's worth a try — especially when you're formatting a whole slew of disks.

D Gookin

The /H switch doesn't exist in PC-DOS before version 3.3. If you're using an MS-DOS version, the easiest way to find out if it has the /H switch is to try it. If it's not implemented in your version of DOS, you'll get the message 'Invalid parameter'.

The batch file QUICKFMT.BAT shown in Fig 1 uses the /H switch to format a

```
ECHO OFF
:LOOP
ECHO N | FORMAT A: /H
IF ERRORLEVEL 1 GOTO NoMore
ECHO PUT a new disk in drive A: now.
ECHO N | FORMAT B: /H
IF ERRORLEVEL 1 GOTO NoMore
ECHO PUT a new disk in drive B: now.
GOTO LOOP
:NoMore
ECHO DONE FORMATTING
```

Fig 1 This DOS 3.3 batch file alternates formatting disks in drives A: and B:

lot of disks, fast. Put blank disks in both drives before you start it. It alternates between drives A: and B:. As soon as one drive finishes, just put another disk in it. The /H switch makes formatting start immediately, and the ECHO N | pipes a 'No' answer to the final 'Format another' question. When you stop putting disks in the drives, the batch file will quit — NR.

Writing quickly with Turbo C

The standard C library provides a number of functions for printing to the screen: printf(), puts() and so forth. I found them slow, however, and wanted a function that would take advantage of the speed of writing directly to the video buffer. The function qwrite() shown in Fig 2 displays a string with a specified attribute to any row and column on the screen by performing a high-speed write to the video screen buffer. It detects the use of a colour or monochrome monitor so that it writes to the correct buffer and waits until a vertical retrace of the screen is in progress to prevent 'snow' on the CGA screen.

Because the qwrite() function accesses the video memory directly, the string can be placed on other screen pages as desired by simply using row values that are greater than 24 (the rows of the first page fall between 0 and 24). The qwrite() function returns the offset of the beginning of the string in the video buffer.

D Winslow

Mr Winslow's qwrite() doesn't really restrict the portability of your programs if you use it wisely. To create a version of qwrite() for other hardware environments, you can substitute the innards of qwrite() with ANSI.SYS escape sequences to position the cursor, and then call printf() in order to print the string. This means you can use qwrite() throughout your programs to do string prints at a specified screen row and column.

There are a couple of interesting points to be noted about how this func-

tion operates. First of all, Mr Winslow uses a far pointer to an integer to access the video buffer; this is useful, since a character and attribute can be modified at the same time. Also, the byte at offset 049h in the BIOS Data Segment (which begins at Segment 040h) determines whether the monochrome or colour video buffer is active.

Finally, two 'while' loops are employed to check the vertical retrace activity on colour monitors. The first loop inspects the vertical retrace bit from the video card to see if a retrace is in progress, forcing the program to stay in the loop until the retrace has ended. Then, the second loop forces the program to wait until the next retrace has begun. By taking this approach, there is usually enough time to update the entire video buffer (4k in size) before the retrace ends.

I've taken the liberty of breaking the original `qwrite()` function into the parts shown in Fig 2. While the `qwrite()` function performs the string-to-screen writes, I've broken the screen mode detection out into a function called `Init_scrptr()`, which sets the far pointer `scrptr` to the appropriate screen buffer. This function should always be called once in your program prior to calling `qwrite()`. It reduces the overhead redundancy of checking for screen mode and resetting the pointer each time `qwrite()` is called. I also placed the video retrace detection in the function `WaitForRetrace()`, which allowed some streamlining of the program logic. In addition, `WaitForRetrace()` can be called by any other function that needs video retrace detection capabilities.

The sample program shown in the list-

ing (Fig 3) demonstrates the use of `qwrite()` by calling it in three loops to write a string in normal characters, then reverse video, and then normal again — RS.

Ready,Set,Go! 4.0

Okay, you graphic-design fiends — get ready (set, go) for this one. Ready,Set, Go! 4.0 can easily wrap text around an imported graphic, right? Well, you can also wrap text *within* two graphics to create interesting paragraph shapes such as circles, triangles — just about any shape your twisted mind can come up with. Here's how to do it.

Draw the shape that you want the text to conform to using a drawing program like MacDraw. The trick is to draw it in two halves (such as two half moons that together would form a circle). Copy and Paste each half of the drawing separately into the Scrapbook.

Now go into Ready,Set,Go! 4.0 and

create two graphic blocks large enough to accommodate your drawings. Butt them up against each other. Paste your two graphics into these graphic frames — the left half into the left block and the right half into the right one. The two halves should now resemble the graphic shape you want. Call up the Specifications box (Command-M) for each graphic block and click Don't Print and Runaround Graphic (as opposed to Runaround Frame).

Next, create one text block as large as the two graphic blocks. Make sure the text block lies directly on top and covers those blocks.

Put your insertion point in the text block and start typing or import a text file. It will take on the shape of the two graphics.

Use justified text if you want the text to conform exactly to the shape you've created. And set text repel distance to zero.

If you have problems getting the text

```
main()
{
    int i,j;

    Init_scrptr();

    for( i = j = 0; i < 24; i++, j++)
        qwrite(i,j,0x7, "This is normal video ");
    getch();
    for( i = j = 0; i < 24; i++, j++)
        qwrite(i,j,0x70,"This is reverse video");
    getch();
    for( i = j = 0; i < 24; i++, j++)
        qwrite(i,j,0x7, "This is normal video ");
}
```

Fig 3 This demonstration program shows the capability of the `qwrite()` function

```
/* qwrite.c
*/
int qwrite(int row, int col, char attrib, char *string)
{
    extern int far *scrptr;
    int far *farptr;
    register int j;

    row += 80; /* reset row for offset into video buffer */
    row += col; /* add column offset to get to first byte */
    if(!string[0]) /* if no bytes to be written, return */
        return row;

    farptr = scrptr + row;
    if( scrptr == (int far *)0xb0000000)
        WaitForRetrace();

    /* put attribute into DH and while not end of string */
    for( j = 0; DH = attrib; string[j]; j++)
    {
        DL = string[j]; /* put character into DL */
        farptr[j] = 0x; /* poke them into the buffer */
    }
    return row;
}

#ifdef OLD
WaitForRetrace()
{
    DX = 0x3da;
    while(( inportb( DX) & 8) != 8); /* get address of video port */
}

#endif
while(( inportb( DX) & 8) == 8); /* wait for next retrace */
#endif
WaitForRetrace()
{
    DX = 0x3da; /* get address of video port */
    while(( inportb( DX) & 8)); /* wait if retrace in progress */
}

int far *scrptr;
Init_scrptr()
{
    if( peekb(0x40, 0x49) == 0x7) /* test for mono */
        scrptr = (int far *)0xb0000000; /* set pointer to mono screen buffer */
    else
        scrptr = (int far *)0xb8000000; /* set pointer to color screen buffer */
}

main()
{
    int i,j;

    Init_scrptr();

    for( i = j = 0; i < 24; i++, j++)
        qwrite(i,j,0x7, "This is normal video ");
    getch();
    for( i = j = 0; i < 24; i++, j++)
        qwrite(i,j,0x70,"This is reverse video");
    getch();
    for( i = j = 0; i < 24; i++, j++)
        qwrite(i,j,0x7, "This is normal video ");
}
```

Fig 2 QWRITE.C provides some primitive tools for access to the video buffer

to flow correctly into the shape, try drawing nonprinting lines above and below your graphic while in RSG! 4.0. This should solve the problem.

How to escape from an endless loop

I really enjoy writing programs in Turbo Basic, but one of its most frustrating limitations is not being able to use the Ctrl-Break combination if my program winds up in an endless loop. To avoid this problem, I have written BREAK.INC (Fig 4), an \$Include file that allows a program to break out of nearly any situation without having to reboot.

The BREAK.INC file should be included at the very beginning of your programs while they are being developed. Then if you get into a situation where the Ctrl-Break combination does not respond, simply press the F10 key instead to break out of the program. Please note, however, that both \$EVENT OFF and KEY(10) OFF will prevent this program from working.

M Wasek

This is a simple and elegant solution to a nasty problem that has always troubled programmers using Turbo Basic. Since the Ctrl-Break combination is ignored by Turbo Basic unless it is performing I/O such as INPUT or PRINT, programmers have traditionally had to reboot to regain control, very often losing any work that was in progress. If your program includes event trapping that also uses the F10 key, you could use any of the other trapable keys instead — EW.

```

----- Break.Inc
ON KEY(10) GOSUB Break
KEY(10) ON
GOTO SkipOver
Break:
END
SkipOver:
    
```

Fig 4 A Turbo Basic \$Include file to stop a program when Ctrl-Break fails

Mac FreeHand

• FreeHand has been criticised for not allowing outlined text to be filled with a shading. But there is, in fact, a way to *simulate* the appearance of filled, outlined text in FreeHand without expending much effort.

First, make sure you've created the shading colour you wish to use. Then select the text tool and create a text

block specifying outlined format. Once you have clicked OK, go immediately to the Edit menu and choose the Clone command. This creates a duplicate of the first text block, right on top of the first one.

Double-click to edit the cloned block. Make sure the text is selected in the resulting dialogue box. Change the text effect box to read 'solid' and change the colour box to be your colour choice (as defined at the beginning). Click OK, then immediately send that text block to the back. This trick works because outlined text in FreeHand is transparent

and lets the shaded text block show through.

C Keegan

• Unless you know how to program in PostScript, the fill and line options in FreeHand can quickly become boring. Expand your options by combining two FreeHand techniques with some ready-made PostScript images — Zapf Dingbats.

To convert any drawn line to a line of dingbats, use the 'text on a path' feature. First draw the line, then choose the Text tool and create a single line of text at a small point size (six point works

well) that is longer than your drawn line. Select both your text block and the line you wish to convert and use the Join Elements command from the Element menu. That's all there is to it.

To use the characters from the Zapf Dingbat font as a fill pattern, use the 'clipping path' feature. Create the area you wish to fill. Select the Text tool and create a block of text large enough to cover your fill area. Then, select the text block and use the Cut command to send it to the Clipboard. Select the area to be 'filled' and choose Paste Inside from the Edit menu.

The same techniques will work with other fonts, but if they aren't PostScript fonts, they will print as bit-mapped characters.

C Keegan

Turbo Pascal noises

A well-written program should be a sight-and-sound experience. However, the in-

ternal procedure Sound() — or even worse, Write(#7) — is nothing short of real boredom. To overcome this, I wrote a small procedure that I now include in each program I write. The result is an easy way to produce all sorts of sounds.

NOISE.PAS takes a handful of integer parameters that describe how to use Sound() and Delay(). Simply code it in as shown in Fig 5 and include the file at the beginning of your source using {\$I NOISE.PAS}. Each time you wish to make weird noises at the user, just say

```
NOISE (b, e, s, d, t, p);
```

where

b = starting frequency in hertz

e = ending frequency in hertz

s = step value from start to end

d = delay in milliseconds between frequency steps

t = how many times the entire sound should be repeated
p = amount of pause in milliseconds between each repeat

NOISE calls can be combined to make all sorts of sounds.

E Kasemodel

I added three more sounds to the demo. It's surprising how many sounds the simple NOISE routine can make. Not every program requires unique sounds, but a distinctive beep can add a certain flair.

NOISE goes through a series of frequencies in steps of fixed size and for a fixed duration at each step. One possible enhancement: add multipliers for both the step and the duration. A multiplier greater than 1 would increase the item on each pass; less than 1 would decrease it. Of course, you'd set the multipliers to 1 for behaviour identical to the existing NOISE program — NR.

```
PROGRAM DEMO for NOISE;
{ Written by E. Kasey Kasemodel }
{ Delete the next line for TP3 }
USES crt;
VAR N : byte;

PROCEDURE Noise(start, (starting frequency)
                stop, (ending frequency)
                step, (step size)
                Del, (time per step)
                Times, (repeats of whole sound)
                Pause (pause between repeats)
                : Integer);
VAR
  a, diff, z : Integer;
BEGIN
  a := Start;
  diff := 0;
  z := 0;
  FOR z := 1 TO Times DO
  BEGIN
    Sound(a); Delay(Del); NoSound; { make sound the first time }
    REPEAT
      IF Start > Stop THEN { noise goes down }
      BEGIN
        a := a-Step; { take step value away from current freq }
        diff := a-Stop; { check difference between freq and stop }
      END
      ELSE { noise goes up }
      BEGIN
        a := a+Step;
        diff := Stop-a;
      END;
      Sound(a); Delay(Del); NoSound; { produce updated sound }
    UNTIL (diff < 0); { keep looping til freq goes past stop }
    a := Start; { start over for another loop }
    Delay(Pause); { wait between loops }
  END; {for z}
END;

BEGIN
  ClrScr;
  WriteLn('NOISE.PAS By E. Kasey Kasemodel');

  Write('01 '); Noise(100, 50, 1, 15, 5, 100); Delay(1000);
  Write('02 '); Noise(100, 250, 10, 50, 3, 100); Delay(1000);
  Write('03 '); Noise(2000, 250, 50, 5, 2, 100); Delay(1000);
  Write('04 '); Noise(50, 2500, 50, 5, 4, 50); Delay(1000);

  Write('05 '); Noise(4000, 1000, 150, 3, 3, 50);
  Noise(1000, 4000, 150, 3, 3, 50); Delay(1000);

  Write('06 '); Noise(1000, 6000, 100, 3, 3, 50);
  Noise(4000, 250, 80, 3, 2, 75);
  Noise(50, 5500, 133, 4, 2, 25);
  Noise(2000, 1000, 60, 3, 3, 50); Delay(1000);

  Write('07 '); Noise(2000, 2400, 2, 2, 2, 50); Delay(1000);

  Write('08 '); FOR N := 1 to 8 DO
  BEGIN
    Noise(1000, 2000, 15, 2, 1, 0);
    Noise(2000, 1000, 15, 2, 1, 0);
  END;
  Delay(1000);

  Write('09 '); Noise(1000, 2000, 500, 2, 200, 0);

END.
```

Fig 5 Use the NOISE procedure to experiment with sound for your programs

Adobe Illustrator

It's easy enough to take an object low in the painting order and bring it in front of everything else in Illustrator on the Mac. But what if you want to bring it forward only so far — as if removing a playing card from a deck and inserting it closer to the top but not right at the top? If you cut the object and paste it in front, then you have to select all the other objects that are supposed to be in front of that object and move them in front to achieve the proper layering. Here's a simpler way.

Select the object you want to bring forward together with the forward object you want to move the first object right behind. Group them while they are both selected (Command-G). That's it. No cutting and pasting. Even if you Ungroup them, the layering change is preserved. The front-most objects do not have to be tampered with.

B Planey

FreeHand

Aldus FreeHand can open Adobe Illustrator 1.1 files for editing, but it can't open Adobe Illustrator 88 files — unless you do the following.

First, open the Illustrator 88 file with Illustrator 88 and Save As an Illustrator 1.1-compatible file to convert custom colours to process colours and defeat masking, as well as remove all patterns and placed or imported images. All objects will be retained. Next, if you have Illustrator 1.1, simply open the file with 1.1 and save it. It can now be opened by FreeHand.

If you don't have Illustrator 1.1, open

the file with a text editor capable of saving text-only files. Change the second line from

```
%%Creator: Adobe Illustrator
88 (TM) 1.6
to
%%Creator: Adobe Illustrator
(TM) 1.1
```

(Note carefully the deletion of the space before **Adobe**.) Save as text only. Open the file with a resource editor and change the Creator from ARTZ to ARTY. The resulting file can be opened and edited in FreeHand.

B Ware

PageMaker 3.0

The new AutoFlow and Text Wrap abilities of PageMaker 3.0 are great, but things can become tricky if you want to wrap text automatically around a graphic, then put a caption below or next to the graphic. Ordinarily, the text being wrapped will flow right over any caption placed below the graphic.

The solution is to increase the 'stand-off' value of the text wrap around the graphic on the side where you want the caption. But there is a trick to this. If you're not careful, the Text Wrap area of the graphic will push the caption away, leaving it stranded right in the middle of the wrapped text again.

You must make sure that all the handles of the text block defining the caption are *inside* the Text Wrap area. To do this, you must place the text or begin the text block by dragging the text cursor, rather than just clicking. Clicking forms a text block the width of the

column or page, which usually makes it larger than the Text Wrap area — the result is that PageMaker will push the caption away. By dragging, you can be sure that the text block of the caption begins and ends inside the wrap area, so it will stay where you want it.

C Aron

Mac Word 3.0

If you plan to type a sizable portion of a document in uppercase text, it is wise to type under the All Caps character format (Command-Shift-K). Not only will you be able to check this text for spelling errors without deselecting the default Ignore Words in All Caps but you will also be able to change the text back to the format of normal lowercase with proper nouns and first words of sentences capitalised (if you still use Shift while typing) simply by removing the All Caps attribute.

B Planey

ImageWriter II

Here's an easy way to coax near-letter-quality printing from the ImageWriter II.

Rather than using ResEdit or Mac Tools (we're not all hackers!) and the Print Merge function of Word 3.0, simply select draft mode when the print dialogue box comes up on the screen. Then, using the selection switches on the printer, select the typeface that you'd like to use in near-letter-quality mode (check your ImageWriter II manual for details on how to select the various choices). It's a snap and works like a charm.

M Silbernik



APC's monthly pot-pourri of hardware and software productivity tips. APC will pay between \$50 and \$200 for each tip published. Send your tip on disk and in printed form to TJ's Workshop, APC, 47 Glenhuntly Road, Elwood 3184.

Activate Ctrl-PrtSc

There are lots of programs that interpret extended ASCII keys and scan codes. However, there aren't very many that will *generate* the extended ASCII keys. TURNPRN.COM is a program that activates Ctrl-PrtSc, the DOS printer echo, and that can be included in any batch file. In order to generate the program, you must first create the DEBUG script file that is shown in Fig 1 with a pure ASCII word processor. Then, from the DOS command prompt, you issue the following command:

```
DEBUG < TURNPRN.DBG
```

TURNPRN.COM simply inserts the Ctrl-PrtSc code directly into the keyboard buffer and resets the tail and head pointers. The code is then read by DOS automatically.

7200 (line six) is the extended ASCII code for Ctrl-PrtSc. The trailing 00 indicates that the key-combination is an extended ASCII code. To use this for other extended codes, change the name of the .COM file and replace 72 with the appropriate code.

D Bell

```
N TURNPRN.COM
A100
MOV AX,40
MOV DS,AX
MOV BX,1E
MOV WORD PTR[BX],7200
MOV BX,1C
MOV BYTE PTR[BX],20
MOV BX,1A
MOV BYTE PTR[BX],1E
INT 20H
```

```
RCX
1A
W
Q
```

Fig 1 This DEBUG script creates TURNPRN.COM, a tiny program that toggles DOS's printer echo on and off

```
ECHO OFF
ECHO This batch file toggles the echo-to-printer feature
ECHO on and off. Be sure your printer is ready, then
PAUSE
TURNPRN
DIR
TURNPRN
ECHO That directory was displayed on your screen AND on
ECHO the printer.
```

Fig 2 TURNDemo.BAT demonstrates the use of TURNPRN.COM. Activating the Ctrl-PrtSc toggle will send anything written to DOS standard output to your screen and printer

When the Ctrl-PrtSc toggle is active, anything that's written to DOS Standard Output goes to the screen and to the printer. You may find this handy in your batch files. I created TURNDemo.BAT (Fig 2) as a minor demonstration of the use of TURNPRN.COM.

When you create the DEBUG script for TURNPRN, be sure to copy it exactly. Don't forget the blank lines before RCX and after the final Q. And be sure to use an editor that can create a flat ASCII file with no control characters and with carriage return and line feed characters at the end of every line. The venerable EDLIN.COM is perfectly suitable for small files like this.

Please note that you probably can't use modifications of the TURNPRN.COM program to bring up hotkey-activated RAM-resident programs. These programs almost invariably check the keyboard port before any information gets put into the keyboard buffer — NR.

SUBST drives

Although many people use the shorthand notation of two periods (..) to switch to a parent directory, most of them fail to realise the enormous usefulness that . and .. can have. These shorthand notations can represent either all of the files in a specified directory or the directory location itself. This flexibility al-

lows . and .. to be used for more than directory changing. For example,

```
SUBST [drive:].
```

This is a neat trick; on a drive with an extensive subdirectory structure it can save you a great deal of time. DOS will substitute the current directory location for the period, and from the point forward, the directory that was current when the command was issued will be assigned a logical drive letter.

Files that appear in two directories can be very quickly and easily compared, copied, and moved by replacing a long path (represented by the period) with a single drive letter.

T Dyck

We've covered the uses of . and .. frequently in this column, but using . with SUBST is new. I've created SUBST-DOT.BAT, shown in Fig 3, that will instantly assign the current directory to a drive letter of your choice. Before you can use SUBST, your CONFIG.SYS must have a line that says

```
LASTDRIVE=n
```

where n is the highest drive letter you want to be able to SUBST.

SUBSTDOT.BAT assumes that you're already using some SUBSTed drives, in this case E: and F:. You wouldn't want

```

ECHO OFF
IF "%1"==" " GOTO NoParam
FOR %v IN (E: F: e: f:) DO IF %1==%v GOTO InUse
SUBST %1 /D > NUL
SUBST %1 .
%1
CLS
ECHO IF you're not in SUBSTed drive "%1" now, either you gave
ECHO an erroneous parameter OR your LASTDRIVE line in CONFIG.SYS
ECHO fails to specify a value of at least "%1".
GOTO End
:NoParam
ECHO SYNTAX: "SUBSTDOT d:", where d: is the drive letter to use.
GOTO End
:InUse
ECHO The drive letter "%1" is in use.
:End
    
```

Fig 3 This batch file will assign the drive letter of your choice to the current directory

to accidentally eliminate your standard SUBST drives, so the batch file checks for those letters and refuses to SUBST them. Of course, you will include whatever letters are appropriate for your own system — NR.

Saving disk space

Most issues of APC have useful DEBUG scripts, which are saved as .SCR and .DOC files. I have found a much better way to save these little files. You can put all the information into a single file and make it very easy to create the corresponding .COM file by writing a batch file as shown below:

```

GOTO batch
{DEBUG script goes here}
{explanations go here}
:batch
DEBUG < %0.BAT > NUL
ECHO %0.COM created
    
```

H Salvisberg

As an example of this idea, I created

WARMBOOT.BAT (see Fig 4). When you run it, it creates WARMBOOT.COM, which reboots your computer.

WARMBOOT is handy in batch files that change CONFIG.SYS and need a reboot to activate the modified configuration.

I added a bit of error checking by piping the output of the DEBUG session through the FIND filter and having it count the occurrences of Error. There will always be one Error message because DEBUG doesn't like the initial GOTO Batch line. That makes it tough to automate error checking. But if more than one error is found, you need to check and correct the script.

You can take any DEBUG script and wrap it up inside a batch file like this. Add the GOTO Batch line at the start and append the lines starting with .Batch to the end. Do note that the .COM file created always has the same name as the batch file, and remember that .COM files have priority over batch files. If you run WARMBOOT once, the batch file creates WARMBOOT.COM. Give the command again and you'll run the .COM file, which reboots your computer — NR.

```

GOTO batch
N WARMBOOT.COM
A 100
MOV BX,1234
MOV AX,0040
MOV DS,AX
MOV [0072],BX
JMP FFFF:0000
    
```

```

RCX
11
W
Q
:Batch
ECHO OFF
DEBUG < %0.BAT | FIND /C "Error"
ECHO IF the line above contains a 1, %0.COM was created
ECHO successfully. Otherwise, check the DEBUG script lines,
ECHO because they contain an error.
    
```

Fig 4 An example of the same file as a batch file and as a DEBUG script

Eliminate ECHO OFF

To suppress ECHO OFF in monochrome, ^[[8m must be the first line of a batch file; this makes the screen invisible (note that ^[represents the ESCape character). An error message is generated but cannot be seen because the screen is invisible. The second line of the batch file is ECHO OFF and ECHO ^[[0m is the third line, which restores the screen.

If you need to eliminate the gap in the appearance of the screen, put the appropriate cursor positioning command in the fourth line. For example, ECHO ^[[3A (the ANSI code to move up three lines) will reposition the cursor up three lines. Of course, the ANSI.SYS driver must be installed, and you will need to look out for case sensitivity in the commands.

If you are working in colour, just switch the foreground colour to the background colour in the first line of your batch file. If you have a blue background, for example, ^[[0;44;34m will send the foreground to blue.

G Feaster

Rejoice! Users of DOS versions less than 3.3 can now avoid the visible ECHO OFF at the start of each batch file. I would add CLS as a fourth line; most of my batch files start with ECHO OFF and CLS as it is.

A tiny editor TED.COM, available from Microtex on Telecom's Viatel on page *6663#, is especially useful for writing this kind of batch file, because you can insert the ESCape character just by pressing the Esc key. You can create a file containing just the three lines described above and simply read this file into the beginning of your other batch files.

Be careful to note that this method is not completely foolproof. If your PROMPT includes ANSI colour commands, the ECHO OFF line will still show up on your screen — NR.

PRINT.COM form feeds

All the printers in my office are installed inside printer mufflers to reduce noise. While this does result in a much quieter office, it also causes the problem of having the last piece of a printout left inside the muffler. Gaining access to this last sheet requires an additional form feed.

The fastest way I've found to cause a form feed is to do a PRINT NUL command. This queues the NUL device to be printed. When PRINT attempts to print NUL, it will advance the paper to

the top of the next form because the NUL device is empty. This is easier than keeping an empty file on your hard disk and PRINTING it.

J Karpinski

Very convenient! The NUL device is always present, and PRINT.COM sends a form feed at the end of every print job, regardless of whether there was anything to print. If you want to, you can put the extra form feeds right in the middle of a particular series of print jobs; for example,

```
PRINT FILE1
PRINT NUL
PRINT FILE2
PRINT NUL
```

That way you'll be able to tear off the printout of each job as soon as it's finished printing — NR.

Replacing a string

Searching and replacing text in a string is easy if the original text is the same length as the replacement. But I needed a more intelligent routine that could substitute multiple occurrences of one string within another, regardless of their

```
SUB SUBST (old$, new$, text$, start)
gap = LEN(old$)
n = LEN(text$)
IF start <= n THEN
  x = INSTR(start, text$, old$)
  DO WHILE x > 0
    text$ = LEFT$(text$, x - 1) + new$ +
    RIGHT$(text$, n + 1 - (x + gap))
    n = LEN(text$)
    x = INSTR(x, text$, old$)
  LOOP
END IF
END SUB
```

Fig 5 A subprogram to replace all occurrences of one string with another

lengths. The program SUBST.BAS, shown in Fig 5, accepts a source string, the strings to search for and to replace with, as well as a starting position in the source string.

J Weisenbach

Because this routine relies on INSTR to do the actual searching, it is case sensitive. However, modifying it to honour capitalisation is easy by using the UCASE\$ function in QuickBASIC 4.0 and Turbo Basic. Simply change the line x = INSTR(start, text\$, old\$) to x = INSTR(start, UCASE\$(text\$), UCASE\$(old\$)). Also, you should specify 1 as a starting offset if the entire source string is to be considered — EW.

Build windows in C

To support the development of programs that incorporate pop-up and pull-down windows in their user interfaces, I have developed a small suite of routines in Turbo C that allow me to save the portion of video memory that will be overwritten by a window, and later close the window by restoring the video buffer.

The functions PushWindow() and PopWindow() work just like push and pop instructions for a stack: you call PushWindow() as many times as needed to open a series of windows, then close them in reverse order with successive calls to PopWindow(). PushWindow() is passed four parameters that define the window's location and size. PopWindow() requires no parameters, it simply restores the screen region whose co-ordinates are next in line on the stack. The function InitWindow() is called once at the beginning to initialise internal variables.

In their present state, these functions furnish bare-bones windowing support. Useful additions might include support for text screens with more than 80 columns, stack overflow and underflow protection, and logic to suppress snow on CGA video adapters.

J Miller

I took you up on your suggestion and re-worked the code slightly, adding support for varying column widths and removing a reference to Turbo C's MK_FP macro to make the code compatible with both Turbo C and Microsoft C. I also added a few lines to include overflow and underflow protection for the global array WinData, which serves as a LIFO stack for window parameters. I assume that when you talk about stack checking, you're referring to WinData and not to the CPU stack. Conventional stack checking is handled by the compiler.

In my modified version of the code shown in Fig 6, PushWindow() and PopWindow() are integer functions whose return values indicate whether or not the call succeeded. Both will return 0 if there is no error. PushWindow() will return 1 if its execution would cause a stack overflow condition, and PopWindow() will return 1 for stack underflow. Their use in the function main() in the listing shows how these two functions should normally be used. In this case, the program simply terminates with an error message if the size of the WinData array (defined by the parameter MAX_WINDOWS) is about to be exceeded.

In addition, PushWindow() returns an

```
#define MAX_WINDOWS 10
#define BUFFER_SIZE 10000

typedef unsigned char BYTE;

struct WindowData{
    BYTE Row;
    BYTE Col;
    BYTE Height;
    BYTE Width;
} WinData[MAX_WINDOWS];

int WinNum = -1; /* Index into WinData array */
unsigned ScreenCols; /* Number of columns displayed */
unsigned far *VideoSeg; /* Pointer to video segment */
unsigned BufferPtr = 0; /* Buffer index */
unsigned WinBuffer[BUFFER_SIZE]; /* Buffer for saved screen data */

main() /* Illustrates the use of window functions */
{
    InitWindow();

    if (PushWindow(0, 0, 8, 40)) {
        printf("Overflow error\n");
        exit(1);
    }

    /*-----*/
    /* Insert code to fill first window here */
    /*-----*/

    if (PushWindow(10, 15, 10, 50)) {
        printf("Overflow error\n");
        exit(1);
    }

    /*-----*/
    /* Insert code to fill second window here */
    /*-----*/

    if (PopWindow()) {
```

Continued . . .

Fig 6 The WINDOWS.C listing shown above contains several routines that support the ability to create pop-up and pull-down windows

```

    printf("Underflow error\n");
    exit(1);
}

if (PopWindow()) {
    printf("Underflow error\n");
    exit(1);
}
}

InitWindow(void)
{
    if (*(unsigned far *) 0x00400063 == 0x3B4)
        VideoSeg = (unsigned far *) 0xB0000000;
    else
        VideoSeg = (unsigned far *) 0xB8000000;
    ScreenCols = *(unsigned far *) 0x0040004A;
}

PushWindow(BYTE Row, BYTE Col, BYTE Height, BYTE Width)
{
    unsigned Offset;
    BYTE i, j;

    if (WinNum == 9) /* Check for overflow */
        return(1);
    if ((BufferPtr + Height * Width) > BUFFER_SIZE)
        return(2);

    WinData[++WinNum].Row = Row; /* Save window parameters */
    WinData[WinNum].Col = Col;
    WinData[WinNum].Height = Height;
    WinData[WinNum].Width = Width;

    Offset = (Row * ScreenCols) + Col;
    for (i=0; i<Height; i++)
        for (j=0; j<Width; j++)
            WinBuffer[BufferPtr++] = *(VideoSeg + Offset +
                (i*ScreenCols) + j);

    return(0);
}

PopWindow(void)
{
    unsigned Offset;
    int i, j;

    if (WinNum == -1) /* Check for underflow */
        return(1);

    Offset = (WinData[WinNum].Row * ScreenCols) + WinData[WinNum].Col;
    for (i=WinData[WinNum].Height-1; i-1; i--)
        for (j=WinData[WinNum].Width-1; j-1; j--)
            *(VideoSeg + Offset + (i*ScreenCols) + j) =
                WinBuffer[--BufferPtr];

    WinNum--;
    return(0);
}

```

Ends

error code of 2 if the capacity of the storage array WinBuffer would be exceeded. The default buffer size of 10,000 words is enough to store the characters and attributes for five windows, each a full 25 lines by 80 columns in size. The number of screen regions that can be stored increases as window size decreases. The default values of 10 for MAX_WINDOWS and 10,000 for BUFFER_SIZE limit total storage capacity to 10 windows or 20,000 bytes, whichever comes first. You can increase available storage space by adjusting these parameters to be proportionately higher.

PushWindow() is called with four parameters: the row and column address of the window's top left corner, the window's height in rows, and its width in columns, in that order. All parameters are zero-based, so the address of the character cell in the upper-left-hand corner of the screen is 0,0. PushWindow() and PopWindow() as-

sume that the current video page is page zero.

The parameters that define a window are stored in a structure. An array of these structures, WinData, holds information on up to 10 windows. When PushWindow() is called to save the contents of a new region, the window parameters passed to it are saved in the array element addressed by WinNum, a variable that keeps track of how many windows have been pushed onto the stack and doubles as an index into the array. PopWindow() retrieves the parameters before restoring a region to its previous state, then decrements WinNum to indicate that there is one less window on the stack.

A record of the current position of the write pointer within the save buffer is maintained in BufferPtr. When character/attribute pairs are transferred to the save buffer by PushWindow(), BufferPtr serves as an index that is incremented after each word is written. PopWindow()

reverses the operation, starting at the current buffer location and reading backwards to restore the window. PushWindow() starts at a window's upper-left-hand corner and works toward the lower right; PopWindow() proceeds in exactly the opposite direction, from lower right to upper left.

Adding snow suppression for CGA video adaptors is more difficult than it might at first seem. The overhead involved in making a function call from a high-level language (and yes, in this context, C is a high-level language) is too great for a program to respond to the signal that a horizontal retrace has begun before the retrace period ends. The solution is to resort to assembly language — a method that presents some powerful alternatives but is beyond the scope of this discussion.

These routines might best be used as part of a larger and more comprehensive library of windowing functions that draw borders, fill windows with text, and more. WINDOWS.C is available for downloading from Microtex on Telecom's Viatel, page *6663# — JP.

MacPaint document affinities

When is MacPaint not MacPaint? It is useful to double-click on a paint file to open it, rather than to load the parent application first. Most clip art comes saved in MacPaint format and will open under MacPaint if double-clicked on. But what if you customarily use FullPaint or SuperPaint instead?

Using a resource editor such as ResEdit or FEdit, or a public-domain utility such as Set File Attributes, you can alter these paint programs so that you can double-click on a MacPaint document and have it load into the paint program of your choice.

Within the resource editor, select the paint program you like to use. Check the field Creator and change what it says to MPNT. If you are using SuperPaint, you also have to do this to the SuperPaint Prefs file. Now quit the resource editor and return to the Finder. You will notice that the icon of your paint program has changed to the MacPaint icon. If the real MacPaint is on disk, get rid of it — otherwise this method won't work. Now double-click on a MacPaint document. It should launch the paint program you patched and load your clip art.

T Asimos

As with all resource-editing tricks, you should always work on a backup copy of your files — Ed.

Group templates, not applications

If you regularly use many applications and want to keep them all handy on your Mac, the usual solution is to group all of them in a single folder on your hard disk. Unfortunately, many applications require that you keep several utilities and sample files in the same folder as the application, which makes spotting the application itself difficult. You can remedy this problem by hiding these utilities outside the visible window or by burying them in folders within the application folder, but there is a better way.

Create an empty document in each of the applications you use regularly and keep it in a folder separate from the application. When using one of these blank documents to create a new file, choose Save As to save the new file and preserve your template. Lock the empty document (using Get Info); this prevents you from making any changes to it.

These blank pages take up little memory and can be customised to your particular needs and serve as style sheets. I use some preformatted and presigned ones for letters and letterheads.

D McLain

Multi-publications in PageMaker 3.01

Have you ever wished you could open more than one publication in PageMaker on the Mac? If you have a lot of memory (at least 3Mbytes), a hard disk and MultiFinder, you can.

Use the Finder to make duplicate copies of PageMaker and rename each copy (eg, PMK.1, PMK.2). For safety's sake, you should probably place each copy in its own folder with accompanying files. Then launch each copy under MultiFinder. Now you can copy and paste whole pages between publications easily.

C Chan

Easy fancy fonts

Can't afford Laser F/X or one of the other font special-effects programs? Here's an insanely simple way to create special effects in Macintosh fonts, even on an ImageWriter.

Take any ordinary font in MacPaint. Type your letters on the screen the way you want them to appear. Then select all the text with the Marquee tool. Now you can either trace the edges with a shadow (Command-Shift-

SPECIAL EFFECTS

SPECIAL EFFECTS

SPECIAL EFFECTS

SPECIAL EFFECTS

Fig 7 Creating special font effects doesn't necessarily require a fancy and expensive program. Even MacPaint has a few tricks — such as these — up its sleeve

E) or trace the edges normally (Command-E). Then you *must* do a Command-Shift-E. Do it several times and experiment with inverting the image and other options to obtain the best results (see Fig 7 for examples).

K Drake

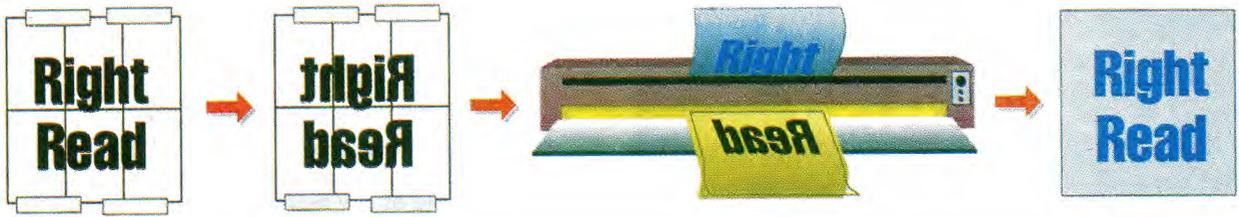


Fig 8 Use MacPaint's Flip Horizontal option to get a reversed image onto a blueprint. Print the flipped image, tape it to the back of the working drawing, and place the two-sided drawing into a blueprint machine. The foreground and background will be correctly superimposed

Using an obtuse option for large drawings

Have you ever wondered what possible practical use Flip Horizontal could have in the LaserWriter Page Setup dialogue box on your Mac? (See Fig 8.) If you need to produce large drawings or Gantt charts on the Macintosh for creating blueprints, you'll find this feature helpful.

You can print out large drawings in 8.5 by 11in sections on a LaserWriter, using the Flip Horizontal option, which creates a mirror image of the original drawing.

The pages can then be trimmed and assembled with transparent tape on the back of the drawing. The resulting 300 dpi flipped-image montage can be used as a master in a blueprint machine.

J Paris

Microsoft Word 3.0

Following up on the tip in the August '88 TJ's column concerning the application of a background screen to a paragraph in Word, here's a PostScript sequence that places a border around an entire page:

```
. page .
/wp$new { newpath 36
36 moveto wp$x 72 sub 0
rlineto 0 wp$y 72 sub
rlineto wp$x 72 sub neg 0
rlineto closepath } def
4.0 setlinewidth wp$new
stroke
```

This sequence places a box with a four-point-wide border indented 36 points from every edge. The number 72 is simply 2 by 36. Change 36 to 72 and 72 to 144 to achieve a 1in border and modify the number before the word `setlinewidth` to suit your needs.

J Wiesenfeld

Apple File Exchange

Here's a tip aimed at people who do a lot of MS-DOS-to-Macintosh conversion, using the Apple File Exchange.

IBM PS/2s and compatibles that use a 3.5in 1.44Mbyte drive can read, write and format a high-density (2Mbyte) disk at the lower 720k density. Unfortunately, when trying to read these disks on a SuperDrive-equipped Macintosh, Apple File Exchange assumes that it is formatted at the 1440k density. AFE is looking at the extra notch on the disk before it attempts to read the actual format.

You can fool the SuperDrive hardware into thinking it has a standard DS/DD (double-sided/double-density) disk by using a piece of tape to cover the extra notch (or to further confuse the issue, you can use a write-protect sticker, as used on those archaic 5.25in floppies).

Also, AFE displays hidden files, such as the desktop, and seemingly transfers them to an MS-DOS disk. However, the only thing that gets transferred is an empty file that is hidden on the MS-DOS side (ie, the file's contents are not trans-

ferred, but a hidden file is created). If you make a habit of hiding your data files under a security program, you'd best unhide them before trying to transfer them to an MS-DOS disk.

C Low

Superior tracing

If you've found yourself disappointed by the results of the auto-trace features of programs such as Illustrator, FreeHand or Canvas 2.0 after the initial excitement, you're in for a pleasant surprise — at least in the case of Canvas.

To make your traces more realistic and closer to the original, first select the bit-mapped object and choose the Object command from Canvas' Object menu. Change the resolution in the dialogue box from 72 to 300 dpi. Click on OK and close the box. Now auto-trace the object. Fig 9 shows the dramatic improvement this step can make. The increased resolution creates more handles, which also makes additional editing easier.

A Alt

END



Fig 9 Canvas 2.0 has an AutoTrace tool with a difference — an undocumented feature. By increasing the resolution of the underlying bit map, using the Object command, you can create better traces. The top object is the original bit map, the middle object is the trace made on a 72 dpi bit map, and the bottom object is a trace made after the same bit map was modified to 300 dpi with the Object command



APC's monthly pot-pourri of hardware and software productivity tips. APC will pay between \$100 and \$200 for each tip published. Write to TJ's Workshop, APC, 124 Castlereagh Street, Sydney 2000.

MacGolf 3.0

If you have Pyro! on your hard disk and accidentally move the cursor to the sleep corner of the screen while playing MacGolf, the screen will black out as usual, but when it comes back on, you'll be faced with a blank screen except for the menu bar.

There is a way, however, to restore the various windows of the game without resetting and starting from scratch.

To restore the Map window on the right side of the screen, first pull down the Options menu and select one of the five View Enlargements that wasn't selected prior to the blackout.

To restore the Golfer in the main window, pull down the Club menu and select a new golf club.

Finally, to restore the background scenery, click the Turn Left or Turn Right button and then the View button.

You're all set — just yell 'Fore!'

R Narusaki

MacPaint

Printing from MacPaint to a LaserWriter is incredibly slow with the new version, (2.0). I've found I can save quite a bit of time by copying the entire screen to the Clipboard, quitting MacPaint, opening Word (or if you're using MultiFinder, switching to Word), pasting the picture

into a new document, and printing it from there.

A Magnori

Customising Mac Excel

Some of the new features in Microsoft Excel 1.5 make it easy to customise, even though there is no Preferences command. I, for example, have a macro sheet that contains all my most commonly used macros. I keep this macro sheet on the top level of my directory and put the actual Excel program in a folder. When I want to start Excel, I simply open this macro sheet. It contains an AUTOEXEC

macro that performs the customisation I require upon startup. This way, the blank worksheet waiting for me is in my favourite font, Palatino 12 point, and the pages will print without grid lines or column heads and with a date stamp instead of a page number. Also, the grid lines are turned off on the display. In addition, a few of my common macros are entered into a couple of custom menus labelled A and B.

By starting a session this way — via the macro sheet and not by double-click-

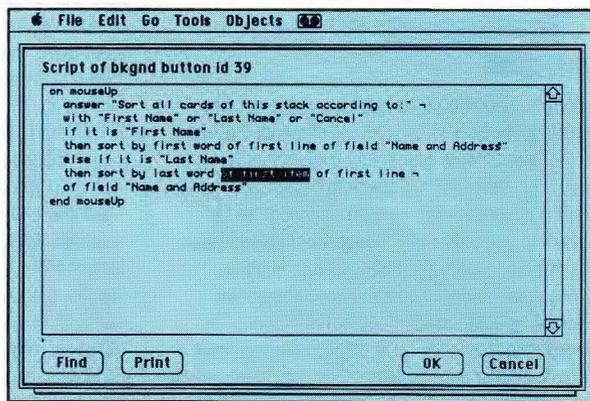


Fig 1 The address stack that accompanies HyperCard does not sort by last names correctly when they include titles like 'Dr.' 'Sr.' or 'Ph.D'. By adding the highlighted words to the script of button ID 39, this problem can be avoided

ing the application icon — not only do I have a customised startup, but I always have my most common macros available to me.

L McArthy

HyperCard sorting fix

A simple addition to the script of the Sort button in the Address stack will allow names followed by academic degrees or 'Jr.' or 'Sr.' to sort correctly by last name when using the 'Last

Name' option on the button. As written, this option causes cards containing the mentioned titles to sort by the title rather than the surname. HyperCard recognises groups of words separated by a comma as an item. Correct usage calls for a comma after the last name and before the added title. Therefore adding the phrase 'of first item' into the script as indicated in Fig 1 results in a correct sort for names either with or without a title.

R Sheavly

Cleaning up screen shots

When you need to isolate an icon or dialogue box in a screen shot to include in a document (see Fig 2a), it's a royal pain to trim off the background grey pattern (usually you have to resort to fat bits to get it exact). Here's a simple way to make editing screen shots easier. Use the Control Panel to change the desktop pattern to white (click on the upper-left arrow until the pattern shows completely white, as in Fig 2b, then click in the tiny window below the arrow to apply the change). Most applications actually use the desktop pattern as their own background pattern, so this change makes screen shots within applications easier too. Now when you take a screen shot (using Command-Shift-3), the screen item you need will be easy to select or lasso all by its lonesome (see Fig 2c).

A Muezza

Keeping path statements short

I like to keep my path as short as possible to minimise directory searches and disk activity. In order to maintain a minimum path, I start programs with batch files that save the current path, change the path to that required by my application, start the application, and then restore the original path. The file SAVEPATH.BAT, shown in Fig 3, is an efficient way to do this. The second line stores the current path in the 'oldpath' environment variable. Lines three and four set up the new path and invoke my application. Lines five and six restore the original path and remove 'oldpath' from the environment.

If you keep the external DOS command programs in C:\DOS and batch files in C:\BAT, the path you set in your AUTOEXEC can be as simple as PATH = C:\DOS;C:\BAT. This can greatly reduce the number of directories and filenames that DOS must search to start an application.

It is a good idea to minimise the use of the APPEND command because of its side effects (see TJ's Workshop, July 1988). However, a minimum APPEND path can also reduce directory searches and disk activity. In order to accomplish this, I start programs requiring an APPEND path with a batch file that saves the old APPEND path, changes it to one appropriate for my application, executes the application, and restores the original APPEND path. The file SAVEAPP.BAT shown in Fig 4 demonstrates how this can be accomplished.

SAVEAPP.BAT works in the following manner. In line two, APPEND with no arguments echoes the APPEND path. If there is none, it echoes 'No Append'. FIND /V filters out the 'No Append' line, so APATH.BAT contains either an APPEND = command or nothing. Line three sets up the appropriate new APPEND path, and line four invokes my application. Line five clears the APPEND path, and line six runs APATH.BAT. If there were a prior APPEND path, the APATH.BAT file restores it; if not, it does nothing.

C Finley

APPEND is a Band-Aid for programs that don't support subdirectories. When you absolutely must use it, the method described here will protect you from its side effects. Note that lines two and six in SAVEAPP.BAT save and restore the previous APPEND path. The safest practice is to have no APPEND path except when it's needed — if you do that you can omit these lines.

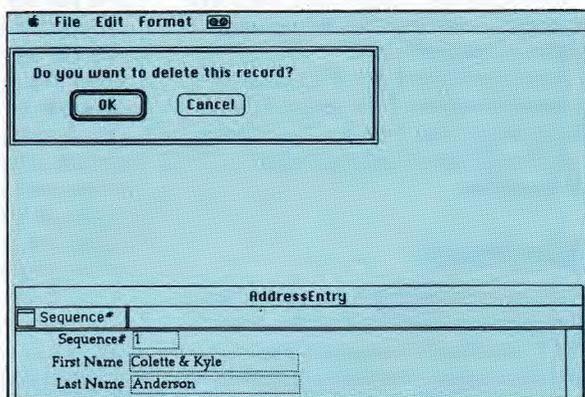
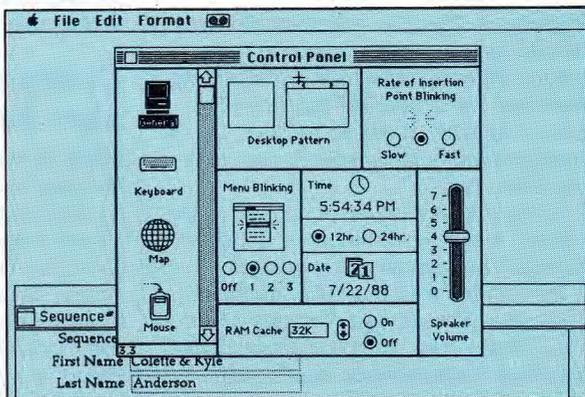
SAVEPATH.BAT wipes out your existing path and replaces it with one specific to the application. Sometimes you may want to keep the existing path and just add another directory to it temporarily. For example, if your application allows you to shell to DOS, you'll want your DOS programs and batch files to be accessible. In that case, just change the third line of SAVEPATH.BAT to

PATH C:\MYAPP;%PATH%

As before, the final lines restore the original path and clear the environment variable 'oldpath' — NR.

Unprotect Basic files

I discovered a method to remove protection from Basic programs written in IBM Basic. I use it because I like to see their



Figs 2a, b and c. Tidying up screen dumps for inclusion in documents can be messy. Clean up your act by learning the ins and outs of the Control Panel. a) shows the typical screen prior to the fateful Command-Shift-3. b) enlightens you to the necessary settings on the Control Panel. c) makes it all whiter than white

listings to learn new things. The first step is to type and run this short program:

```
10 DEF SEG
20 BSAVE "UNPROBAS", 1124, 1
RUN
```

LOAD the program you want to unprotect, and then BLOAD the file UNPROBAS. That unprotects the file. Now LIST the program to see its contents, or SAVE it unprotected.

H Ali

Every now and then, this tip comes up again; it's a shame they don't just put it in the manual. When you save a Basic file with the P option, it sets a flag in the program's header that tells Basic not to LIST or SAVE the program. BLOADING the UNPROBAS file overwrites that flag. Note that once you've created UNPROBAS, you can just store it with your Basic files and use it as needed.

Here are the exact instructions to type into the Basic interpreter if you want to unprotect a file called SECRET.BAS:

```
LOAD "SECRET"
BLOAD "UNPROBAS"
```

```
SAVE "SECRET"
```

That does it — BW.

Accenting paragraphs

The following sequence of PostScript commands will accent a paragraph with a background screen in Mac Word:

```
.para. gsave .97 setgray
wp$box fill grestore
```

The grey intensity is set to .97 (where 1 represents white and 0 black). The space above and below the paragraph (as set in the Paragraph menu) is included in the shaded region and this background screen resizes itself if the paragraph is edited.

D Sarwate

Detecting a DOS shell

Many programs provide the ability to exit temporarily to a secondary copy of COMMAND.COM. Although there is usually an initial indication that you are in a secondary command shell, it quickly scrolls off the screen.

One solution I have found is to invoke

```
ECHO OFF
SET OLDPATH=%PATH%
PATH C:\MYAPP
MYAPP %1
PATH %OLDPATH%
SET OLDPATH=
```

Fig 3 This batch file sets up an application-specific path, then restores the original path

```
ECHO OFF
APPEND | FIND /V "No Append" > c:\APATH.BAT
APPEND C:\MYAP\MYDATA
C:\MYAP\MYAP
APPEND:
C:\APATH.BAT
```

Fig 4 A batch file that protects you from the side effects of the APPEND command

such programs with a batch file that includes the following commands:

```
SET PROM=%PROMPT%
SET PROMPT=[%0]%PROMPT%
{other commands to invoke
program}
SET PROMPT=%PROM%
```

TJ'S WORKSHOP

The first two SET commands save the current value of the DOS prompt in the environment variable 'prom' and prefix the prompt with the name of the batch file. The last SET command restores the prompt to its original setting. Now if I use the DOS shell facility of the program invoked by the batch file, I have a clear indication that this is not the primary copy of COMMAND.COM.

The batch file SETPROMP.BAT shown in Fig 5 demonstrates this technique. This batch file invokes a secondary copy of COMMAND.COM, which will remain the active command processor until it receives an EXIT command. Until the secondary COMMAND.COM exits, the DOS prompt will be prefixed with [SETPROMP] (assuming you name the batch file SETPROMP.BAT). When you EXIT the secondary shell, the prompt will return to its normal assignment.

M Reibstein

It's easy to forget that you're running in a shell — it looks just like DOS. Some programs (WordPerfect, for example) already provide a reminder, but for those that don't, this is a nice solution. If you prefer, you can include a more explicit message than the bracketed batch file

```
ECHO OFF
SET PROM=%PROMPT%
SET PROMPT=[%0]%PROMPT%
ECHO EXIT to return to the primary COMMAND.COM
COMMAND
SET PROMPT=%PROM%
SET PROM=
```

Fig 5 This batch file demonstrates how the prompt can remind you that you're running in a secondary command processor

name. For example, replace the third line of SETPROMP.BAT with

```
SET PROMPT=[EXIT to
return]%PROMPT%
```

Try running SETPROMP.BAT several times in a row. Each time you run it you invoke another secondary COMMAND.COM, and each time your prompt grows. To get back to the main command processor, just keep entering EXIT until your prompt returns to normal — NR.

Grabbing italics on a Mac screen

Setting the insertion point and selecting

individual letters can be difficult and inexact when working with italics — partly because the insertion point is vertical and italic letters are not and also because italic screen fonts are usually terrible. Here is a simple way to set the insertion point and select individual letters consistently — no more hit and miss:

1. Move the I-beam to the *centre* of the letter *just in front* of the letter that you want to select.

2. Click. The insertion point will be set immediately in front of the desired letter.

That's it. Now you can be certain that any typing at this point will be placed where you want it. You can also easily drag from the insertion point to the right and always select the desired letter. Select the first letter of a line of italics by

clicking well to the left of the line and dragging to the right until a selection 'pops' up on the screen.

P Gaines

More Turbo Pascal lines and columns

There are some situations in which the standard IBM PC video screen (80 by 25) is not big enough to display all the information we want to show on a single screen — for example, when you need to display a summary table, a correlation matrix, etc. In those situations, you may wish you could have a monitor that can display 150 columns and 40 or 50 lines.

One way out of this limitation is to create a virtual screen. The virtual screen can hold all the information you want and at any time display a part of it on the video monitor. Your screen just becomes a 'viewport' into the virtual screen that you can drag around using the arrow keys, as shown in Fig 6. The program listing in Fig 7 provides the basic procedures to do just that.

The `Big_ClrScr` routine is similar to the Turbo Pascal built-in function `ClrScr`. It clears the entire virtual screen (not the

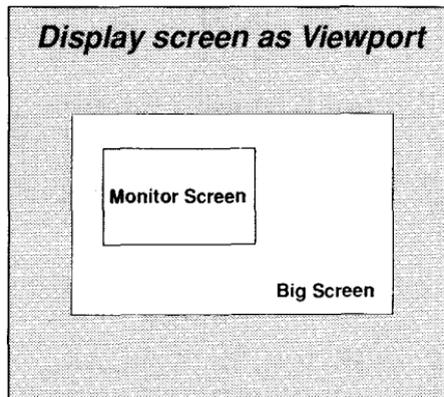


Fig 6 A visual representation of the video monitor as a viewport into a larger virtual screen

monitor screen). The `Big_Write` procedure puts the text given in the first parameter on the virtual screen. The three next parameters specify the position on the screen (Col, Row) and the video attribute of the text (VidAtt). The `Show_BigScreen` routine moves a part of the virtual screen on the video monitor. The two parameters specify the upper-left corner of the window on the virtual screen. This procedure is called by the `GetMove` procedure that waits for

a key from the keyboard. If you press an arrow key, the viewport moves in that direction.

N Peladeau

Be warned — this program will cause ugly 'snow' on a standard IBM CGA. Accessing the video RAM directly while the electron beam is writing to the screen causes this snow. If you adopt this technique, you will need a procedure to access video RAM during the retrace interval, when the electron beam is sweeping back for the next scan line. Almost every library of Turbo Pascal routines includes such a procedure. In addition, the procedure `Show_BigScr` checks the video mode by directly examining a byte in the BIOS data area. This technique generally works, but it is safer for you to simply query the video interrupt, as in the program listing `VID-MODE.PAS` that is shown in Fig 9.

Cursor shapes in Turbo Pascal

Many programs that modify the cursor will return the cursor as a dash instead of an underline when run on a monochrome system. This happens

```

PROGRAM Bigger Screen;
USES CRT, TURBO3;
(* NOTE: Delete the line above for use in Turbo Pascal 3.0. The
program works as is in TP4. *)
CONST
Wide = 150; { You can adjust the size of the virtual screen }
Long = 40; { by changing these two constants }
Nline = 25; { change to 24 if your monitor displays only 24 lines }

TYPE
Scr = ARRAY[1..Nline, 1..80, 1..2] OF Char;
LongString = STRING[255];

VAR
MonOD : Scr ABSOLUTE $B000 : $000;
ColOD : Scr ABSOLUTE $B000 : $000;
Mode : Byte ABSOLUTE $0040 : $0045; { video mode }
BigScr : ARRAY[1..Long, 1..Wide, 1..2] OF Char;

PROCEDURE Big_ClrScr;
VAR I : Integer;
BEGIN
FOR I := 1 TO Wide DO
BEGIN
BigScr[1, I, 1] := Chr(32);
BigScr[1, I, 2] := Chr(15);
END;
FOR I := 2 TO Long DO Move(BigScr[1, 1, 1], BigScr[I, 1, 1], Wide*2)
END;

PROCEDURE Big_Write(Line : LongString; Col, Row : Integer; VidAtt : Byte);
VAR I : Integer;
BEGIN
IF (Row <= Long) THEN
FOR I := 0 TO Length(Line)-1 DO
IF (Col+I) <= Wide THEN
BEGIN
BigScr[Row, Col+I, 1] := Line[i+1];
BigScr[Row, Col+I, 2] := Chr(VidAtt);
END;
END;

PROCEDURE Show_BigScr(col, Row : Integer);
VAR I : Integer;
BEGIN
FOR I := 1 TO Nline DO
CASE Mode OF
7 : Move(BigScr[i+col-1, row, 1], MonOD[i, 1, 1], 150);
2, 3 : Move(BigScr[i+col-1, row, 1], ColOD[i, 1, 1], 160);
END;
END;

PROCEDURE GetMove;
VAR
Ch1, Ch2 : Char;
PcsX, PosY : Integer;
BEGIN
PosX := 1; PosY := 1;

ClrScr;
REPEAT
Show_BigScr(PosX, PosY);
Read(Kbd, Ch1);
IF Ch1 = Chr(27) THEN
BEGIN
Read(Kbd, Ch2);
CASE Ord(Ch2) OF
72 : IF PosX > 1 THEN PosX := Pred(PosX); {up }
77 : IF PosY < Wide-79 THEN PosY := Succ(PosY); {right}
80 : IF PosX < Long-Pred(Nline) THEN
PosX := Succ(PosX); {down }
75 : IF PosY > 1 THEN PosY := Pred(PosY); {left }
END;
UNTIL Ch1 = Chr(13);
END;

BEGIN
Big_ClrScr;
Big_Write('<- this is the upper left corner of the screen',
1, 1, White);
Big_Write('Use arrow keys to move the "viewport"; press <Return> to quit',
1, 2, White);
Big_Write('<- this is the middle of the screen',
Wide DIV 2, Long DIV 2, Red);
Big_Write('this is the lower right corner of the screen ->',
Wide-46, Long, 25);
GetMove;
END.

```

Fig 7 A program demonstrating a virtual screen larger than the monitor

```

(SR+)
PROGRAM Cursor;
VAR
  I, J, X : Integer;
  Equipment_Flag : Integer ABSOLUTE $40:$10;
PROCEDURE Set_Cursor(Top, Bottom : Byte);
TYPE
  Registers = Record
    CASE Integer Of
      1 : ( AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer );
      2 : ( AL,AH,BL,BH,CL,CH,DL,DH : Byte );
    END;
VAR
  recpack : Registers;
BEGIN
  Recpack.AH := 1;
  Recpack.CH := Top;
  Recpack.CL := Bottom;
  Intr($10, recpack);      (call interrupt)
                           ( Set_Cursor )
END;

CASE ParamCount OF
  0 : IF Equipment_Flag AND $30 = $30 THEN ( Monochrome board? )
      Set_Cursor(11, 12)
    ELSE
      BEGIN
        WriteLn('Auto-reset is for MONO system only.');
```

```

      WriteLn('Use 2 numeric parameters (to set ',
        'cursor top and bottom, e.g. "Cursor 2 10").');
```

Fig 8 The program CURSOR.PAS lets you set the cursor shape in Turbo Pascal

when the program saves the original cursor shape and resets it to that shape when it's done.

A simple solution to this problem is to set the cursor manually before running any cursor-modifying programs. Doing so also corrects the cursor scan line numbers that are stored in memory. The program shown in Fig 8 does just this if run with no parameters. It first checks to see that a monochrome board is in use. You can include the program in your AUTOEXEC.BAT file if you like. If you call it with two parameters, it will set the cursor's top scan line to the first and the bottom to the second. Creative parameter juggling can yield cursors that look like blocks, underbars, or even double blocks. You can change the cursor to let the user know what is going on, for example, whether he is in insert or overwrite mode. (Note: Fig 8 was written in Turbo Pascal 3.0. To convert it to 4.0, use the compiler's UPGRADE.EXE

utility.) One interesting quirk I've discovered is that while many parameter sets simply yield no cursor, others such as

CURSOR 100 105

give a cursor that looks normal but blinks very slowly.

E Woodhouse

The parameter set

CURSOR 0 12

would set a full block on a monochrome monitor. For a CGA, you'd use

CURSOR 6 7

The slow-blinking or erratic cursor is a phenomenon of the monochrome monitor only. Note that this program uses a somewhat chancy method of

checking the current video mode. It looks at the BIOS Equipment Flag for the monitor currently in use. In general, you should query the video interrupt to get the current mode, as in the listing VIDMODE.PAS in Fig 9 — NR.

Turbo Pascal RAM size check

It can be handy for your program to know how much memory its host computer has. When an application may run on a variety of different hardware, you can use this information to set up a balance of memory and disk-based storage. The function MemSize included in Fig 10, which was written in Turbo Pascal 3.0, calls interrupt \$12 to get that information. The number returned by MemSize is the amount of memory installed in the system, not the amount of free RAM.

Kevin King

```

(SR+)
PROGRAM VidModeDemo;
VAR
  VidMode : Byte;
FUNCTION VideoMode : Byte;
CONST
  BiosVidSvc = $10;      (* Bios Video Services Interrupt *)
  GetVidMode = $0F;     (* Bios Request for Get Video Mode *)
TYPE
  RegRec = RECORD
    AX, BX, CX, DX : Integer;
    BP, SI, DI      : Integer;
    DS, ES, Flags  : Integer;
  END;
VAR
  Registers : RegRec;
BEGIN
  Registers.AX := GetVidMode SHL 8;
  Intr(biosVidSvc, Registers);
  VideoMode := Lo(Registers.AX);
END;
END; (* VideoMode *)
BEGIN
  VidMode := VideoMode;
  IF VidMode = 7 THEN
    WriteLn('Using MONOCHROME monitor, 80x25 TEXT mode');
  ELSE
    BEGIN
      Write('Using Color/Graphics monitor, ');
      CASE VidMode OF
        0 : WriteLn('40x25 black-and-white TEXT mode');
        1 : WriteLn('40x25 color TEXT mode');
        2 : WriteLn('80x25 black-and-white TEXT mode');
        3 : WriteLn('80x25 color TEXT mode');
        4..6 : WriteLn('CGA graphics mode ',VidMode);
        8..10 : WriteLn('PCjr graphics mode ',VidMode);
        11,12 : WriteLn('UNKNOWN graphics mode ',VidMode);
        13..16 : WriteLn('EGA graphics mode ',VidMode);
      END;
    END;
  END;
END.
```

Fig 9 The program listing for VIDMODE.PAS is a routine to check the current video mode

```

($R+)
PROGRAM MemSizeTest;

FUNCTION MemSize : Integer;
TYPE
  Regs = RECORD
    AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer;
  END;
VAR Registers : Regs;
BEGIN
  Intr($12, Registers);
  MemSize := Registers.AX;
END;

FUNCTION KAvail : Integer;
  { We multiply the high and low bytes of MemAvail }
  { separately in order to avoid problems when its }
  { value is > 32767. Turbo will treat an integer }
  { > 32767 as a negative number, but bytes are }
  { always positive. }
  BEGIN
    KAvail := Trunc((Hi(MemAvail)*256.0+Lo(MemAvail))/64.0);
  END;
BEGIN
  WriteLn('TOTAL RAM installed is ', MemSize, 'K. ');
  WriteLn('TOTAL RAM available within this program is ', KAvail, 'K. ');
END.

```

Fig 10 A program that checks the TOTAL amount of installed RAM

Turbo's own MemAvail function tells how many 16-byte paragraphs of RAM your program can use beyond its basic code and data requirements. If your program needs a lot of RAM, you can check what's available with a function like KAvail in Fig 10.

Using MemSize, you can deliver an intelligent message if KAvail shows too little RAM: for example, 'This program needs 400k to run. You have 640k installed, but only 200k of it is available to me. Please remove some RAM-resident programs and try again'. (Note: In order to convert Fig 10 to Turbo Pascal 4.0, use the compiler's UPGRADE.EXE utility.) — NR.

Red Ryder

In the commercial versions of Red Ryder (10.0 and later), Scott Watson chose not to document a useful feature that was sketchily documented in the 9.0 Read-me file. This feature concerns how to include your own icon in your own menu. You do this in conjunction with the ADD TO MENU procedure command. To do this, you use a caret (^) character after the command, followed by a number. There are two steps involved:

1. Create an icon in a copy of the Red Ryder application using ResEdit. The icon should be an ICON resource and

should have an ID number 265 or higher. Close ResEdit and save your changes.

2. Include a line like this in your procedure file:

```
ADD TO MENU ^9Genie log-on<S
```

See Fig 12 for an entire procedure file. Refer to the Red Ryder manual for information on how to create a menu and how to use the ADD TO MENU command.

In this example, ^9 stands for the icon with ID number 265. Menu icons are numbered starting with 257, which means ^1 refers to the icon with ID num-

ber 257, ^2 refers to 258. Things get even weirder when you get past ^9. You can't enter ^10 because there must be a single digit after the caret. Instead, you have to follow the ASCII character set, where the colon (:) follows the character 9. So if you create an icon with ID number 266, you would enter ^: to refer to it.

That's it! Now when you run your procedure, Red Ryder inserts the icon you created to the left of the menu item (see Fig 13).

D Valiulis

QuickBASIC alert

There's a particularly nasty bug in QuickBASIC 3.0 that causes some programs to crash. If a program calls assembler routines and it is compiled to a BCOM file from within the editor, a faulty .OBJ module will be created. One solution is to always compile from DOS, but the best move is to trade up to QuickBASIC 4.0.

Underline fix

The Underline style in Mac Word results in an underline that clings too closely to the word — the line slices right through any descenders. Sometimes the word and underline begin to resemble black scrambled threads rather than emphasised text. Here's how to unscramble this mess, using Word's formula mode.

Select Show ¶ from the Edit menu. Press Command-Option-⌘ to enter formula mode. Type an O to select the overstrike command, an opening parenthesis, the words to be underlined, a comma, a number of spaces equivalent to the length of your text, and then a closing parenthesis. It should now look like this on screen:

```
\O (peppery pungency, .....)
```

Now select all the spaces following the

comma and choose Character from the Format menu. Click Underline and Subscript. Type in the number of points you want the underline subscripted. You'll have to experiment, since the ideal amount differs from font to font and depends on the point size of the text.

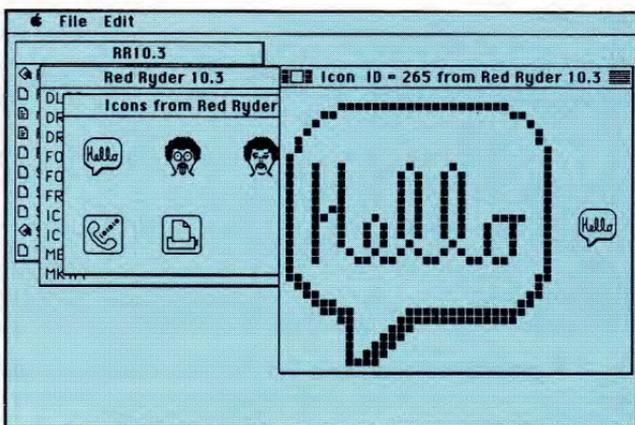


Fig 11 Customised menus with user-created icons can be created in Red Ryder — a feature not mentioned in the documentation of the commercial version. Here an icon is forged in ResEdit and added to Red Ryder

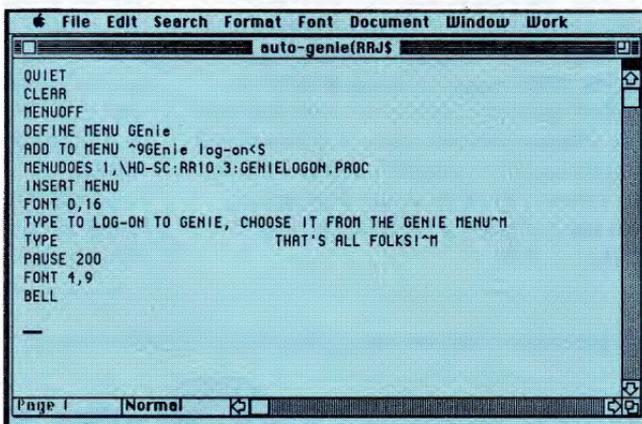


Fig 12 A sample procedure file in Red Ryder showing how the icon is added

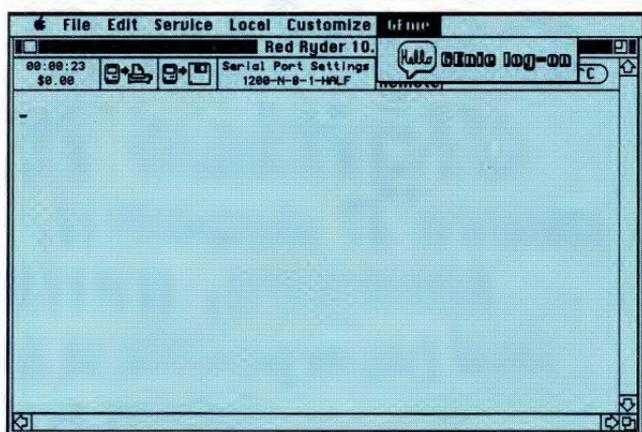


Fig 13 Hello there! Your icon greets you when the menu is pulled down

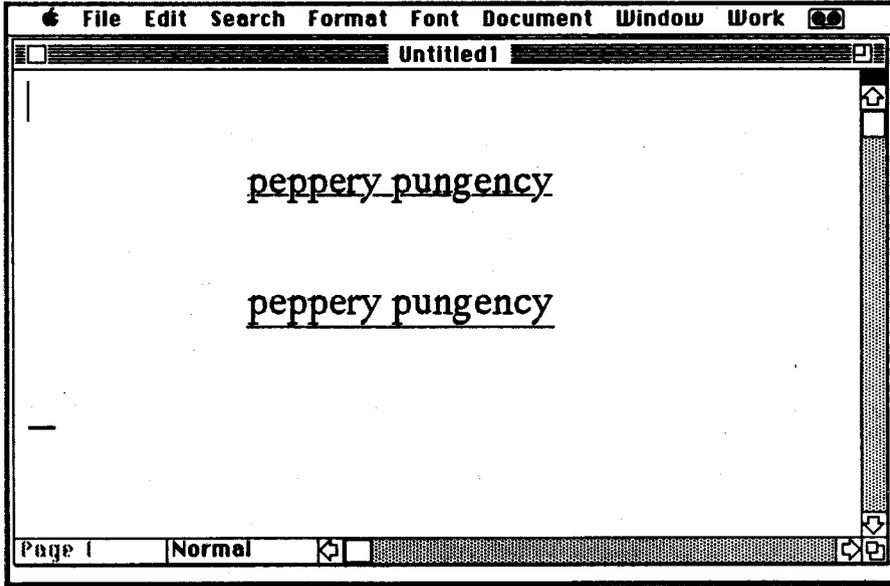


Fig 14 The underline style in most word processors is less than ideal in appearance. Word's formula mode provides a workaround. The top words are underlined using the usual command from the Format menu. The lower words are underlined in formula mode

Click OK and select Hide ¶ from the Edit menu. The formula coding will vanish and your underlined word or words will appear. See Fig 14 for examples.

The subscripting might change the leading (line spacing) of the line with the underlined text. To avoid this, enter a specific number for line spacing in the Paragraph dialogue box rather than the usual Auto. If the problem persists, insert a minus sign in front of the line-

```

N ANSI.COM
A 100
MOV     SI,0082
MOV     CL,[SI-02]
JCXZ   0117
DEC     CX
LODSB
CMP     AL,7E
JNZ    0110
MOV     AL,1B
XCHG   DX,AX
MOV     AH,02
INT    21
LOOP   0109
RET

RCX
18
W
Q
    
```

Fig 15 This DEBUG script generates ANSI.COM, a tool to communicate with ANSI.SYS

spacing number. This forces Word to keep the leading at the indicated amount no matter what.

You should save a sample like this to Word's glossary. Then when you want to use this alternative to the inferior underline style on the Format menu, just insert it and substitute new text and the appropriate number of spaces.

V Swanson

Enhance your DOS environment

The earliest computer monitors were little more than electronic teletype machines. And, just like a teletype, they'd print one character at a time until the end of the line, then advance to the start of the next line. In this age of pop-up windows and pull-down menus, that teletype action seems primitive, but it's the way most DOS commands and many utilities work.

Later video terminals added control codes. These codes move the cursor, erase areas of the screen, and generally control the display. You can add this kind of control to your batch files and DOS environment using ANSI.SYS.

Before you try to install ANSI.SYS, make sure it is in the root directory of your boot disk. Then add this line to your CONFIG.SYS file:

```
DEVICE = ANSI.SYS
```

Now reboot your computer. You won't

notice any immediate difference, but ANSI.SYS is at your command.

To send ANSI.SYS a message, you write an escape sequence to standard output. An escape sequence is a coded string of characters beginning with ASCII character 27, the Esc character. Here we run into a problem: it's difficult to ECHO the Esc character. Yet, to experiment with ANSI.SYS you need an easy way to send it messages. To this end I'm providing ANSI.COM. This tiny program accepts a character string on its command line, replaces every tilde (~) character with an Esc, and writes it to standard output. For example,

```

ANSI ~[2J           clears the
screen
ANSI ~[H           homes the
cursor
ANSI ~[41;33;1m    sets yellow
text
on red background
    
```

To create ANSI.COM, type in ANSI.SCR as shown in Fig 15 using any editor that can generate flat ASCII files. Make sure you include the blank lines after the RET instruction and after the final Q. With DEBUG.COM in the current directory or available on the path, give the command

```
DEBUG < ANSI.SCR
```

Watch the output — if you see '^ Error' pointing to a line, double-check that line in ANSI.SCR, fix it, and repeat. Try your

ANSI Colour Commands		
Parameter	Effect	
0	All off (low-intensity white on black)	
1	High intensity	
4	Underline on (monochrome only)	
5	Blink on	
7	Reverse video	
8	Invisible	
Colour	Foreground	Background
Black	30	40
Red	31	41
Green	32	42
Yellow	33	43
Blue	34	44
Magenta	35	45
Cyan	36	46
White	37	47

Fig 16 ANSI colour commands take the form ESC[#; . . . ;#m, where each # represents a number from the list above

ANSI Cursor Commands

Cursor direction	Code
Backward	ESC[#D
Forward	ESC[#C
Down	ESC[#B
Up	ESC[#A
To row/column	ESC[#;#H
To row/column	ESC[#;#I
Clear screen	ESC[2J
Erase to EOL	ESC[K
Save cursor position	ESC[s
Restore saved position	ESC[u

Fig 17 The # symbol represents an optional parameter that determines how many times the command will be repeated. For example, ESC[3C moves the cursor forward three characters. It defaults to 1

new ANSI.COM with the examples above or create your own. You can use any ANSI sequence as long as you replace the Esc with a tilde.

You won't find the ANSI commands in your DOS manual unless you're using an old version of DOS. Figs 16 and 17 provide a brief listing of the handiest commands. ANSIBOX.BAT, shown in Fig 18, demonstrates what you can do with these ANSI commands in a batch file. The first ANSI command prints 'X marks the spot (X)' and saves the current cursor position. The next three commands write a yellow-on-red box with yellow-on-black text inside it. Then the last line comes back to the previous cursor position, turns off all colour attributes, and finishes that line. With ANSI.COM you can beautify all your batch files — NR.

Executing repetitive operations

To perform repeated operations with a single keystroke combination, I created ANSIKEY.BAT shown in Fig 19. To assign a text string to a key combination, enter

```
ANSIKEY F# [text of key setting, up to 8 words].
```

```
ECHO OFF
ANSI X marks the spot (X)~[s
ANSI ~[5;27H~[41;33;1m
ANSI ~[6;27H~[40m HERE IS TEXT IN A BOX ~[41m
ANSI ~[7;27H
ANSI ~[u~[0m... and here we are back at the spot.
```

Fig 18 An example of how to use ANSI to beautify batch files

```
ECHO OFF
IF NOT "%9"==" GOTO TooMany
GOTO %1
GOTO ERROR
:F1
ANSI ~[0;84;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F2
ANSI ~[0;85;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F3
ANSI ~[0;86;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F4
ANSI ~[0;87;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F5
ANSI ~[0;88;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F6
ANSI ~[0;89;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F7
ANSI ~[0;90;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F8
ANSI ~[0;91;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F9
ANSI ~[0;92;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:F10
ANSI ~[0;93;"%2 %3 %4 %5 %6 %7 %8";13p
GOTO OK
:TooMany
ECHO Key setting text can contain at most 8 words.
GOTO DONE
:ERROR
ECHO SYNTAX: "ANSIKEY F# [text of key setting, up to 8 words]"
GOTO DONE
:OK
ECHO %1 %2 %3 %4 %5 %6 %7 %8 > \%1.KEY
:DONE
```

Fig 19 ANSIKEY.BAT makes assigning text strings to keys easy

For example,

```
ANSIKEY F1 CD\DOS\DBASE\DRAWINGS
```

will set function key F1 to execute

```
CD\DOS\DBASE\DRAWINGS
```

when you enter Shift-F1.

To store the key settings, I create ten files, F1.KEY through F10.KEY. List the key assignments stored in these files by

using ANSIKEYL.BAT in Fig 20.

The .KEY files must be initialised at startup via ANSIKEYI.BAT, shown in Fig 21, which I execute within AUTOEXEC.BAT.

All three batch files — ANSIKEY.BAT, ANSIKEYL.BAT, and ANSIKEYI.BAT — require that ANSI.SYS be installed. To do this, move ANSI.SYS into your root directory and add this line to your CONFIG.SYS:

```
DEVICE = ANSI.SYS
```

Within ANSIKEY.BAT, I use ANSI.COM to send the appropriate command to ANSI.SYS. For function keys F1 through F10, I send the scan codes 84 through to 93.

This same technique could be used to assign values to other keystroke combinations. I have chosen Shift-F# because it seems to be a key combination that's less used than Alt or Ctrl, for ex-

```
ECHO OFF
ECHO Key assignments for shifted function keys
REM "COPY ... CON > NUL" hides any "File not found" messages.
FOR %%f IN (1 2 3 4 5 6 7 8 9 10) DO COPY \F%%f.KEY CON > NUL
```

Fig 20 ANSIKEYL.BAT lists the current key assignments made by ANSIKEY.BAT

ample. This technique saves me a substantial amount of time, and it uses only standard MS-DOS commands.

P Hayes

The submitted ANSIKEY.BAT file relied on the PROMPT command to send key redefinition strings to ANSI.SYS. As a result, it wiped out the existing prompt. To save the prompt, I fixed AN-SIKEY so it uses the ANSI.COM program instead.

Any command you create with AN-SIKEY will automatically have a carriage return added to it. If you don't want that carriage return, remove the three characters ;13 from each ANSI command line.

ANSIKEY allows you only to assign text strings to shifted function keys. If you'd like to experiment further, you can use ANSI.COM to create other key reassignments. The syntax is

```
ESC[#;#;..#P
ESC[#;"string"p
ESC[#;"string";#;"string"p
```

The first number (#) is the ASCII code of the key you're redefining. If this number is 0, the second number is the extended ASCII code of the key. The remainder of the command is the character sequence you want to assign to that key. This sequence can contain any number of quoted text strings and ASCII character numbers separated by semicolons. For example,

```
ANSI ~[0;84;"DIR *.COM
/W";13;"DIR *.EXE /W";13p
```

will set the Shift-F1 key to do first a wide directory of .COM files and then a wide directory of .EXE files.

ANSI key reassignments work only in programs that use DOS standard input

and output. There aren't many such programs these days, but the DOS command level itself, along with utilities such as DEBUG and EDLIN, are likely places to use them — NR.

Boxing with formulae

How do you put a box within a box in Mac Word 3.0X? The program's formula mode rides to the rescue once again.

Ordinarily you box a paragraph or

```
ECHO OFF
ECHO F1 > \F1.KEY
ECHO F2 > \F2.KEY
ECHO F3 > \F3.KEY
ECHO F4 > \F4.KEY
ECHO F5 > \F5.KEY
ECHO F6 > \F6.KEY
ECHO F7 > \F7.KEY
ECHO F8 > \F8.KEY
ECHO F9 > \F9.KEY
ECHO F10 > \F10.KEY
```

Fig 21 ANSIKEYI.BAT initialises the .KEY files that track the current key assignments

```

DEF FNGetKey$
'Works like INKEY$, but also returns a repeat count for extended keys.
STATIC GetKey$, ScanCode$, Mask$, Count$, InArray$(1), OutArray$(1)
REDIM InArray$(7), OutArray$(7)

GetKey$ = INKEY$           'get a keystroke
FNGetKey$ = GetKey$       'set up default return value
IF LEN(GetKey$) <> 2 THEN EXIT DEF 'if null or normal key, exit

ScanCode$ = ASC(RIGHT$(GetKey$, 1)) 'isolate the key's scan code
Mask$ = 8                  'assume it's an Alt-key for now
Count$ = 1                'assume no repeats for now

SELECT CASE ScanCode$
CASE 94 TO 103             '<ctrl-F1> through <ctrl-F10>
  Mask$ = 4
CASE 115 TO 119           '<ctrl-Left>, <ctrl-Right>, <ctrl-End>,
  Mask$ = 4                '<ctrl-PgDn>, or <ctrl-Home>
CASE 132                  '<ctrl-FgUp>
  Mask$ = 4
CASE ELSE
END SELECT

'Wait for additional keystrokes until the user either releases the
'Alt or <Ctrl> key, or hits a different keystroke.
L1: InArray$(0) = $H200

'Call the BIOS to get the status of the <ALT> and <CTRL> keys.
CALL INT86($H16, VARPTR(InArray$(0)), VARPTR(OutArray$(0)))

'Check the appropriate bit in the AL register.
IF (Mask$ AND OutArray$(0)) = 0 GOTO Done

```

```

'Call the BIOS to see if there is a keystroke waiting in the buffer.
InArray$(0) = $H100
CALL INT86($H16, VARPTR(InArray$(0)), VARPTR(OutArray$(0)))

'Check the 2 flag in the FLAGS register to see if a keystroke is waiting.
IF (OutArray$(7) AND 64) = 64 GOTO L1

'Examine AH register to see if keystroke matches the previous scan code.
IF PEEK(VARPTR(OutArray$(0)) + 1) <> ScanCode$ GOTO Done

'Remove the keystroke from the keyboard buffer and increment the count.
GetKey$ = INKEY$
Count$ = Count$ + 1
GOTO L1

Done: IF Count$ <> 1 THEN FNGetKey$ = GetKey$ + CHR$(Count$)
END DEF

'Test program for FNGetKey$
L2: Key$ = FNGetKey$
IF Key$ = "" GOTO L2

SELECT CASE LEN(Key$)
CASE 1
  PRINT Key$;
CASE 2
  PRINT "Scan Code: "; ASC(MID$(Key$, 2));
CASE 3
  PRINT "Scan Code: "; ASC(MID$(Key$, 2));
  PRINT "Count: "; ASC(RIGHT$(Key$, 1));
END SELECT
PRINT
GOTO L2

```

Fig 22 A QuickBASIC function that distinguishes multiple keystrokes

group of selected paragraphs by selecting Box from the Border selections in the Paragraph dialogue box. Box choices are Single, Thick, Double, or Shadow.

To do a box within a boxed group of paragraphs, first turn Show ¶ on. Press Command-Option-\ and then an X. Type the text to be boxed with parentheses after the X. Then Hide ¶, and an inner box will appear. The nested box is limited to a single line of text, and any return, including soft returns, will cancel

the box formula command.

You can use this feature to create boxed labels within a boxed table. The boxed text can be bold or italicised — any variation normally available from the Format menu or the Character dialogue box.

V Swanson

Multiple keystroke recognition

Programs that use PgUp and PgDn can

be improved if you can move by more than one page at a time. This is often done by combining PgUp and PgDn with Ctrl to page in higher, fixed increments.

A better alternative to this is to count the number of times that the keys have been pressed while the Ctrl key is held down. This creates keystrokes such as Ctrl-PgUp-PgUp-PgUp, which can be interpreted to mean 'Page up three pages'. This could also be applied to graphics software that has a zoom

```

PROGRAM Copy;
CONST
  BufSize = 16384;
VAR
  ioCode          : Byte;
  SrcFile, DstFile : FILE;
  FileNameA, FileNameB : STRING[255];
  Buffer           : ARRAY[1..BufSize] OF Byte;
  RecsRead        : Integer;
  DiskFull        : Boolean;

PROCEDURE SrcFileError(ioCode : Byte);
BEGIN
  Write($7, 'I/O result of ', ioCode, ' (decimal) ', #26);
  CASE ioCode OF
    $01 : WriteLn(' Source file not found. ');
    $F3 : WriteLn(' Too many files open. ');
  ELSE WriteLn(' "Reset" unknown I/O error. ');
  END;
END;

PROCEDURE DstFileError(ioCode : Byte);
BEGIN
  Write($7, 'I/O result of ', ioCode, ' (decimal) ', #26);
  CASE ioCode OF
    $F0 : WriteLn(' Disk data area full. ');
    $F1 : WriteLn(' Disk directory full. ');
    $F3 : WriteLn(' Too many files open. ');
  ELSE WriteLn(' "rewrite" unknown I/O error. ');
  END;
END;

BEGIN
  Write('Copy from file : ');
  ReadLn(FileNameA);
  Write('      To file : ');
  ReadLn(FileNameB);
  IF FileNameB <> FileNameA THEN
    BEGIN
      Assign(SrcFile, FileNameA);
      Assign(DstFile, FileNameB);
      (* Note second parameter in "reset" and "rewrite" of untyped files. *)
      {$I-} Reset(SrcFile, 1); {$I+}

```

```

      ioCode := IOResult;
      IF ioCode <> 0 THEN SrcFileError(ioCode)
      ELSE
        BEGIN
          {$I-} Rewrite(DstFile, 1); {$I+}
          ioCode := IOResult;
          IF ioCode <> 0 THEN DstFileError(ioCode)
          ELSE
            BEGIN
              DiskFull := False;
              WHILE (NOT Eof(SrcFile)) AND (NOT DiskFull) DO
                BEGIN
                  (* Note fourth parameter in "blockread". *)
                  {$I-}
                  BlockRead(SrcFile, Buffer, BufSize, RecsRead);
                  {$I+}
                  ioCode := IOResult;
                  IF ioCode <> 0 THEN
                    BEGIN
                      SrcFileError(ioCode);
                      DiskFull := True;
                    END
                  ELSE
                    BEGIN
                      {$I-}
                      BlockWrite(DstFile, Buffer, RecsRead);
                      {$I+}
                      ioCode := IOResult;
                      IF ioCode <> 0 THEN
                        BEGIN
                          DstFileError(ioCode);
                          DiskFull := True;
                        END
                      END
                    END
                END;
              IF NOT DiskFull THEN WriteLn('File copied. ');
            END;
          Close(DstFile);
        END;
      ELSE WriteLn($7, 'File can not be copied onto itself. ');
    END;

```

Fig 23 An improved version of the COPY.PAS program with built-in error checking

```

PROGRAM Delete;
VAR
  ioCode : Byte;
  FileVar : FILE;
  FileName : STRING[255];
PROCEDURE ioError(ioCode : Byte);
BEGIN
  Write(#7, 'I/O result of ', ioCode, ' (decimal) ', #26, ' ');
  CASE ioCode OF
    $01 : WriteLn('Filename not found.');
```

```

    $20 : WriteLn('Illegal operation for a logical device.');
```

```

    $F3 : WriteLn('Too many files open.');
```

```

    ELSE WriteLn('Unknown I/O error.');
```

```

  END;
END;
BEGIN
  Write('Delete file : ');
  ReadLn(FileName);
  Assign(FileVar, FileName);
  {$I-} Erase(FileVar);      {$I+}
  ioCode := IOResult;
  IF ioCode <> 0 THEN ioError(ioCode)
  ELSE WriteLn('File deleted.')
```

```

END.
```

Fig 24 A better way to delete any file without leaving the Turbo editor

capability. If Alt-Z means 'Zoom two times', then Alt-Z-Z could mean 'Zoom four times', and Alt-Z-Z-Z could mean 'Zoom eight times', and so forth.

FNGetKey\$ is a QuickBASIC function that implements this scheme (Fig 22). It works like INKEY\$, but it returns an extra character containing a repeat count for selected combinations of keystrokes using Ctrl or Alt.

J Parsly

As with Inkey\$, you would use the length of the string returned to determine if the key that was pressed was normal, extended, or repeated. Some text editors, such as SideKick, use a similar technique to avoid repeatedly updating the screen.

If you press and hold Ctrl-PgUp in SideKick, it senses that you want to span several screens and waits until you stop key pressing. Then, it redraws the screen only once.

This function can be converted to work with Turbo Basic without much effort. Turbo Basic supports calling DOS and BIOS interrupts directly — RH.

Word counting

Here's an easy way to obtain an estimated word count for a Mac Word document.

Make sure the insertion point is at the top of the document. Access Change from the Search menu. Type in ^32 in the Find What box, and ^32 in the Change to box. Or simply press the space bar in each box. Click Change All. In a few moments the number of changes made appears in the lower left of the window. This yields a good approximation of the number of words in the document. And since you are changing white spaces into white spaces, you are, in effect, not making any changes at all.

You run no risk of harming all your work. The only problem arises with long files: if you are short on memory, Word may not be able to complete the procedure. Also, if your words are separated by tabs or returns (as in a long list of words), this technique won't work well.

J Mowreader

Copying and deleting in Turbo Pascal

Mr David Johns gave examples of some useful programs in TJ's Workshop, May 1987. His Copy and Delete programs worked well, but I found them inadequate. I've developed versions with run-time error checking and a display of what occurred during their execution. The programs are listed in Figs 23 and 24.

When I want to use one of these programs, I \$Include it at the top of the program I'm working on, for example, {\$IC:\TURBO\COPY.PAS}. Including the program is much cleaner than copying it into my current program and deleting it when I'm finished. It's easy to remove the single \$Include line.

R Christopher

Clever! Turbo Pascal stops compiling when it reaches a legitimate 'END.' statement. e.g. if it's in an \$Include file. Hence you can \$Include a whole program at the top of the program you're working on — NR.

```

{$R+}
PROGRAM GetASCII;
CONST
  bufsize = 16384;
TYPE
  BuffType = ARRAY[1..bufsize] OF Char;
VAR
  FilVar : FILE;
  Filename : STRING[64];
  Actual : Integer;
  MinLen : Byte;
  buffer : BuffType;
  ASCII : ARRAY[Char] OF Boolean;
  overlap : Byte;
PROCEDURE Check(VAR B : BuffType; top : Integer; VAR ovLap : Byte);
VAR P, Pend, I : Integer;
BEGIN
  P := 1;
  ovLap := 0;
  REPEAT
    IF ASCII[B[P]] THEN
      BEGIN
        Pend := P;
        REPEAT
          Pend := Succ(Pend)
        UNTIL (NOT ASCII[B[Pend]]) OR (Pend = top);
        IF ((Pend = top) AND ASCII[B[Pend]]) THEN
          ovLap := Succ(Pend-P)
        ELSE
          IF Succ(Pend-P) >= MinLen THEN
            BEGIN
              FOR I := P TO Pred(Pend) DO Write(B[I]);
              WriteLn;
```

```

      END;
      P := Succ(Pend);
    END
  UNTIL (P >= top);
END;
END;
BEGIN
  FillChar(ASCII, SizeOf(ASCII), False);
  FillChar(ASCII[#32], 96, True);
  (* Now for any character CH, ASCII[CH] is true *)
  (* only if CH is a character from #32 to #127 *)
  Write('Enter minimum string length to check for : ');
  ReadLn(MinLen);
  Write('Enter name of file to check: ');
  ReadLn(Filename);
  Assign(FilVar, Filename);
  Reset(FilVar, 1);
  WHILE NOT Eof(FilVar) DO
    BEGIN
      BlockRead(FilVar, buffer, bufsize, Actual);
      Check(buffer, Actual, overlap);
      (* Back up to get any that "cross" into the next buffer *)
      IF (Actual = bufsize) AND (overlap > 0) THEN
        Seek(FilVar, FilePos(FilVar)-overlap);
      (* NOTE: This is a Turbo Pascal 4.0 program, but you can
      convert it for TP3 by deleting the line above
      and replacing it with the line below. *)
      (* LongSeek(FilVar, LongFilePos(FilVar)-overlap); *)
      END;
      Close(FilVar);
    END.
```

Fig 25 An easy way to find ASCII strings in a .COM or .EXE file

Finding ASCII text in .EXE or .COM files in Turbo Pascal

Here is a simple Turbo Pascal program, listed in Fig 25, which you can use to find any ASCII text in an .EXE or a .COM file. I use it to see all the messages a program prints to the screen and to look through programs obtained from bulletin boards for any messages that may indicate a 'bomb' program (although this is not always a foolproof method).

M McGuffey

The original program simply printed out every printable character in the source file. I enhanced it a bit to let you choose a minimum length for the strings you'll see. The program puts each string of at least the minimum length on a separate line. If you see 'Arf! Arf! Gotcha!' in the output, think twice before running the program. Note that the program doesn't check for existence of the input file — bad filenames will crash it — NR.

Ready,Set,Go! 4.0

Ready,Set,Go! 4.0 has no way to

generate a table of contents (TOC) automatically. Here's a fairly simple way to create a TOC that will stay current even as you change the document.

Create a text block on the page where you want the table of contents. Be sure that the block is only large enough to hold the one entry and the page number. Enter the text for the entry and then press Command-Option-Shift-7 where you want the page number to go. The symbol '&&' will appear, which is RSG's way of saying 'text continues on' — something mentioned only on page 201 of the 'Reference Shortcuts' documentation. Create similar blocks for each TOC entry.

Now use the linking tool to link each TOC entry to the target page text block. The TOC entry should now show the page number of the article or chapter. And it will be updated even if you add or delete pages.

This technique requires that you carefully pre-plan your document and create approximate text blocks for each item to be included in the table of contents. But it will still save you time and a lot of headaches.

J Buono