See Next Page

SHELL Shines in DOS Window

An assembly-language routine lets you SHELL to a predefined window for maximum control of your DOS user interface.

Tony Elliott is Vice President and General Manager of ElfTech Development Inc. and author of several add-ons for Basic, including F-Tree Plus, Printer Plus, Compression Plus, and Fax Plus. He can be contacted at 4574 Shallowford Industrial Parkway, Marietta, Georgia 30066, or on CompuServe at 76220,2575.

If you're like most programmers, you create polished user interfaces. You make sure that the colors are well balanced and pleasing to the eye, and that anything the user needs to do is only a keypress or mouse click away. It's frustrating to lose control of your meticulously handcrafted screen displays when you SHELL out to a DOS (or other) command-line utility. During the SHELL, these utilities can cause your display to wrap around and scroll, temporarily destroying all of your hard work. Wouldn't it be great if you could confine the output of such utilities to a predefined window on the screen during a SHELL? After hearing several programmers complain about this problem, I created a solution. The DosWindow routine described here lets you control the screen display during a SHELL and add a first-class professional touch to your applications.

Before I discuss how this routine works, consider the organization of operating system software and ROM BIOS services. There are at least three software layers in every PC: BIOS, DOS, and application programs.

At the lowest level, there's the system hardware. This includes all the IC chips and circuit boards that make up your computer. Generally speaking, there can be literally hundreds of

```
; DOSWIN.ASM - Listing 1
; DOS windowing routine for use during a SHELL
; Copyright (C) 1993 ElfTech Development, Inc.
; Written by Tony Elliott - August 1993
; Requires MASM 6.0x or later to reassemble. This routine
; is compatible with QB/PDS/VBDOS, and PowerBasic 3.0x.
.Model Medium, Basic
.Code
;                Variable Declaration Area
;; We're going to store our local variables in the code
; segment in order to avoid any impact on DGroup.

OrigInt10       LABEL DWord     ;A "far" label pointing to
OrigInt10Ofs    dw 0            ; interrupt 10h's original
OrigInt10Seg    dw 0            ; vector.
OrigInt29       LABEL DWord     ;A "far" label pointing to
OrigInt29Ofs    dw 0            ; interrupt 29h's original
OrigInt29Seg    dw 0            ; vector.
TopRow          db 0            ;When the programmer
LeftCol         db 0            ; defines the window
BottomRow       db 0            ; coordinates and color
RightCol        db 0            ; attribute, they are
ColorAttr       db 0            ; stored here.
LineWrap        db 0
ScrnRows        db 25           ;The current display
ScrnCols        db 0            ; parameters will be
ScrnPage        db 0            ; stored here.
;                       Define Macros
VideoInt Macro                  ;Some BIOS can destroy
    Push    bp                  ; the BP register. Using
    Int     10h                 ; this macro, we make sure
    Pop     bp                  ; that BP is always
Endm                            ; preserved.
;                     Procedure Area
DosWindow proc uses DS, TopR:Word, LeftC:Word,
        BottomR:Word, RightC:Word, Colr:Word, LWrap:Word
; On entry to this routine, let's get some information
; about the current video adapter.
    Cmp     cs:OrigInt10Seg,0   ;Are we already installed?
    Jz      @f                  ;If not, continue
    Call    RemoveDosWindow     ;If so, deinstall first
@@: Push    cs
    Pop     ds                  ;Point ds to code segment
  Assume    ds:@Code
    Mov     ah,0fh              ;Get video mode
```

(CONTINUED ON NEXT PAGE)

Hooking an Interrupt. This complete assembly-language routine allows you to create a predefined window while SHELLing from QB, PDS, VBDOS, or PB. Making this routine work requires intercepting all calls made to interrupt &H29 and &H10 during the SHELL.

machine-language instructions required just to put a single character on your video display. And to make matters more complicated, these steps can vary from one hardware configuration to another. Realizing this early on, IBM created the specifications for a group of software subroutines called Basic Input Output System, or BIOS.

The BIOS specification defines the interrupt numbers and parameters used to control everything from disk I/O and printer services to keyboard handling and screen display. Manufacturers that want to produce PC-compatible motherboards, video cards, and other types of hardware must also include a set of BIOS functions, usually stored in ROM or a device driver, that conform to the IBM specifications. These functions act as the software layer between the operating system and the hardware. For example, if I want to switch the current video adapter into the 80-by-25 color mode from assembly language, I don't need to know all the low-level steps required by the specific video card I'm using; I simply use BIOS interrupt &H10 service. The video card manufacturer has provided a group of interrupt &H10 functions designed specifically for his card. This approach makes life much easier.

The software layer above the BIOS is the operating system: DOS. DOS provides memory management, file and directory organization, program loading and termination services, and much more. For example, the BIOS provides services for reading and writing individual sectors on a hard disk, and DOS provides the higher-level organization that keeps track of where on disk the files are stored, and so forth. In other words, when manipulation of hardware devices is necessary, DOS functions will usually call BIOS routines instead of manipulating the hardware directly. This approach prevents DOS from becoming dependent on one specific brand of hardware. The software layer above DOS is an application program. Application programs can use DOS and even BIOS services to handle keyboard input, screen I/O, file I/O, and other operations.

Knowing the sequence of events that transpire when an application prints a string allows us to write a routine like DosWindow. For example, when most DOS command-line utilities want to print a character, they use the DOS "fast put character" function (interrupt &H29). This function simply writes a character to the screen at the current cursor location and moves the cursor to the next character position. It also processes the carriage return (ASCII 13) and line feed (ASCII 15) characters, and manipulates the cursor accordingly when these characters are encountered. Interrupt &H29 doesn't access the video hardware directly. It makes calls to the video BIOS interrupt &H10 services to get the current cursor location, print the character, move the cursor to the next position, and to scroll the screen when necessary.

In order to make the DosWindow routine work, you must intercept all calls made to

LISTING 1. (CONTINUED FROM PREVIOUS PAGE.)

```
    VideoInt                        :Call the video BIOS
    Dec     ah                      :Make screen cols base 0
    Mov     ScrnCols,ah             :Screen columns
    Mov     ScrnPage,bh             ;Active display page
    Xor     ax,ax
    Mov     es,ax                   :Point es to BIOS data
    Mov     al,byte ptr es:[484h]   :Rows, if EGA/VGA
    Cmp     al,42                   :43 line mode? (base 0)
    Jz      RowsOk                  :If so, continue
    Cmp     al,49                   :50 line mode?
    Jz      RowsOk                  :If so, continue
    Mov     al,24                   ;Assume 25 line mode
RowsOk:
    Mov     ScrnRows,al             :Store rows in our var
'   Now, load and evaluate the window coordinates provided
'   by the Basic program.
    Mov     ax,TopR                 :TopRow% into ax
    Dec     al                      :Make it base 0
    Cmp     al,ScrnRows             :Greater than ScrnRows?
    Ja      BadParam                :Exit if so
    Mov     bx,LeftC                :LeftCol% into bx
    Dec     bl                      :Make it base 0
    Cmp     bl,ScrnCols             ;Greater than ScrnCols?
    Ja      BadParam
    Mov     cx,BottomR              :BottomRow% into cx
    Dec     cl                      :Make it base 0
    Cmp     cl,al                   :Less than TopRow%?
    Jb      BadParam                :If so, exit
    Cmp     cl,ScrnRows             :Greater than ScrnRows?
    Ja      BadParam                :If so, exit
    Mov     dx,RightC               :RightCol% into dx
    Dec     dl                      :Make it base 0
    Cmp     dl,bl                   :Less than LeftCol%?
    Jb      BadParam                :If so, exit
    Cmp     dl,ScrnCols             :Greater than ScrnCols?
    Ja      BadParam                :If so, exit
'   All of the coordinates seem to be ok. Load them into
'   our internal variables along with the remaining
'   two parameters.
    Mov     TopRow,al
    Mov     LeftCol,bl
    Mov     BottomRow,cl
    Mov     RightCol,dl
    Mov     ax,Clr                  :Color attribute into ax
    Mov     ColorAttr,al            :Store it
    Mov     ax,LWrap                :LineWrap% into ax
    Mov     LineWrap,al             :Store it
'   All of the parameters are now stored in our internal
'   variables. Let's now clear the window area using the
'   specified color and position the cursor to the upper-
'   left corner of the window.
    Mov     ax,600h                 :Scroll up 0 lines (clear)
    Mov     bh,ColorAttr            :Color attribute into bh
    Mov     ch,TopRow               :Window coordinates into
    Mov     cl,LeftCol              ;CX and DX.
    Mov     dh,BottomRow
    Mov     dl,RightCol
    VideoInt                        :Call the video BIOS
    Mov     ah,2                    :Set cursor position to
    Mov     bh,ScrnPage             : the upper-left corner
    Mov     dh,TopRow               ; of our window.
    Mov     dl,LeftCol
    VideoInt
'   The window is clear and the cursor is positioned. Now
'   get interrupt 10h's current vector and store it. Then
'   point int 10h to our handler so we can take control.
    Mov     ax,3510h                :Get the current vector
    Int     21h                     : for interrupt 10h.
    Mov     OrigInt10Seg,es         ;Store it for later
    Mov     OrigInt10Ofs,bx
    Mov     ax,3529h                :Get current vector for
    Int     21h                     : interrupt 29h.
    Mov     OrigInt29Seg,es         ;Store it for later
    Mov     OrigInt29Ofs,bx
    Mov     ax,2510h                :Point the int 10h vector
    Mov     dx,offset OurInt10      ; to our code
    Int     21h
    Mov     ax,2529h                ;Point Int 29h to our code
    Mov     dx,offset OurInt29
```

(CONTINUED ON NEXT PAGE.)

[redacted] [redacted]

Listing 1. (Continued from previous page.)

```
    Int     21h
    Xor     ax,ax                   ;Return 0 (successful)
DosWindowExit:
    Ret                             ;Back to Basic
BadParam:                           ;If we detected a bad
    Mov     ax,-1                   ; parameter, return -1.
    jmp     short DosWindowExit
DosWindow endp
;                   Our Interrupt 10h Handler
; When an interrupt 10h hits, we check to see if it is
the
; "scroll zero" function. This is used to clear a
; rectangular area of the screen (or the entire screen).
; DOS and command line utilities use this service to
; perform a CLS operation. If a scroll zero request is
; detected, we redefine the coordinates to include only
; our window. Anything else, we pass right through to
; the original int 10h handler.
OurInt10 proc far
    Assume  ds:Nothing, cs:@code
    Sti                             ;Enable interrupts
    Push    ds                      ;Preserve ds
    Cmp     ax,600h                 ;A scroll up 0? (CLS)
    jz      ScrollZero              ;If so, process it
    cmp     ah,9                    ;Multiple char TTY?
    jz      CharPrint
    cmp     ah,0eh                  ;Or single char TTY
    jz      CharPrint               ;Process them
    Jmp     ChainToOrigInt10        ;If no of above, chain
CharPrint:                          ;If function 9 or E
    mov     bl,cs:ColorAttr         ; use this attribute.
    Jmp     ChainToOrigInt10        ;Chain to orig int 10h

ScrollZero:                         ;If a scroll zero
    Push    cs                      ;
    Pop     ds                      ;Point ds to code
  Assume    ds:@code
    Mov     ch,TopRow               ;Override the requested
    Mov     cl,LeftCol              ; CLS area (whole screen)
    Mov     dh,BottomRow            ; with the coordinates
    Mov     dl,RightCol             ; and color defined for
    Mov     bh,ColorAttr            ; our window.
ChainToOrigInt10:
    Pop     ds                      ;Restore ds
    Jmp     Dword Ptr cs:OrigInt10  ;Chain orig int 10h
OurInt10 endp
;                   Our Interrupt 29h Handler
; When an interrupt 29h (DOS "Fast PutChar") function
; hits, we check the cursor position and make sure it is
; within our window. If it is outside the window, we
; either wrap to the next line, or scroll the window up.
OurInt29 proc far
    Sti                             ;Turn interrupts back on
    Push    ds                      ;Save these registers
    Push    ax
    Push    bx
    Push    cx
    Push    dx
    Push    es
    Pop     ds                      ;Point ds to code segment
  Assume    ds:@Code
    Mov     ah,3                    ;Get current cursor pos
    Mov     bh,ScrnPage             ;Returned in DH/DL
    VideoInt
;   As we qualify the output, BH and BL are used as
;   control flags. If BH is set, then the current
;   character is not printed (CR, BL, or if past right
;   border of window when the LineWrap flag is clear).
;   BL is set when the cursor position needs to be
;   updated (line wrap or carriage return).
    Xor     bx,bx                   ;Clear our control flag
CheckCR:
    cmp     al,0dh                  ;A carriage return?
    jnz     CheckLF                 ;If not, continue
    Mov     dl,LeftCol              ;If so, set left col to
    Mov     bx,-1                   ; left side of window and
    jmp     short CheckBounds       ; set "update cursor
flag"
CheckLF:
    cmp     al,0ah                  ;Is it a line feed?
    jnz     CheckBounds             ;If not, continue
    mov     bx,-1                   ;If so, set both flags
    Inc     dh                      ;Point to the next line
CheckBounds:                        ;Check cursor bounds
    Cmp     dl,LeftCol              ;Position < LeftCol?
    Jae     CheckRightCol           ;If not, continue
    Mov     bl,-1                   ;Set "reset cursor" flag
    Mov     dl,LeftCol              ;New column for cursor
    Jmp     short CheckRow          ;No need to check
RightCol
CheckRightCol:
    Cmp     dl,RightCol             ;Position > RightCol?
    Jbe     CheckRow                ;If not, continue
    Cmp     LineWrap,0
    jnz     @f
    Mov     bh,-1
    jmp     short CheckRow
@@: Mov     bl,-1                   ;Set cursor pos flag
    Mov     dl,LeftCol              ;Cursor back to LeftCol
    Inc     dh                      ;And move to next row
CheckRow:
    Cmp     dh,TopRow               ;Above TopRow?
    Jae     CheckBottomRow          ;If not, continue
    Mov     bl,-1                   ;Set cursor repos flag
    Mov     dh,TopRow               ;New row to set cursor to
    Jmp     short FixUp             ;Don't check BottomRow
CheckBottomRow:
    Cmp     dh,BottomRow            ;Below BottomRow?
    Jbe     FixUp                   ;If not, continue
    Mov     bl,-1                   ;Reset cursor flag
    Mov     dh,BottomRow
;   If we get here, we need to scroll the window up a
line
    Push    bx                      ;Save our control flags
    Push    dx                      ;And new cursor position
    Mov     ax,601h                 ;Scroll up one line
    Mov     bh,ColorAttr            ;Load registers with
    Mov     ch,TopRow               ; coordinates of
    Mov     cl,LeftCol              ; window and color
    Mov     dh,BottomRow            ; attribute.
    Mov     dl,RightCol
    VideoInt
    Pop     dx                      ;Restore the registers
    Pop     bx
FixUp:                              ;Adjust cursor pos, if
    Push    bx                      ; needed
    Or      bl,bl                   ;Reset the cursor pos?
    Jz      @f                      ;If not, continue
    Mov     ah,2
    Mov     bh,ScrnPage
    VideoInt
@@: Pop     bx                      ;Restore flags again
    Or      bh,bh                   ;Skip printing this char?
    Pop     dx
    Pop     cx
    Pop     bx                      ;Restore these registers
    Pop     ax
    Pop     ds
    jnz     @f                      ;If skip print, return
    Jmp     DWord Ptr cs:OrigInt29  ;Otherwise, chain
@@: Iret
OurInt29 endp
;.....
RemoveDosWindow proc uses ds
;   From Basic:
;     DECLARE SUB RemoveDosWindow ()
;   Unhooks the DosWindow routine from int 10h and resets
;   our internal variables.
    Push    cs
    Pop     ds                      ;Point ds to code
  Assume    ds:@Code
    Cmp     OrigInt10Seg,0          ;Are we installed?
    Jz      RemoveExit              ;If not, exit
    Mov     ax,2510h                ;Set vector for int 10h
    Mov     dx,OrigInt100fs         ;Offset of orig handler
    Mov     bx,OrigInt10Seg         ;Seg of handler in bx
    Mov     OrigInt10Seg,0          ;Reset our "flag"
```

(Continued on next page.)

LISTING 1. (CONTINUED FROM PREVIOUS PAGE.

```
    Mov     ds,bx                   ;Seg of orig handler in ds
    Assume  ds:nothing
    Int     21h                     ;Call DOS
    Mov     ax,2529h                ;Restore original int 29h
    Mov     dx,cs:OrigInt29Ofs
    Mov     bx,cs:OrigInt29Seg
    Mov     ds,bx
    Int     21h
RemoveExit:
    Ret                             ;We're finished
RemoveDosWindow endp
end
```

```
'TESTWIN.BAS - LISTING 2
DEFINT A-Z

DECLARE SUB RemoveDosWindow()
DECLARE FUNCTION DosWindow% (BYVAL TopRow%, _
                    BYVAL LeftCol%, BYVAL BottomRow%, _
                     BYVAL RightCol%, BYVAL Colr%, _
                     BYVAL LWrap%)
'PB3 users, uncrem the following line:
'$link "dosbox2.obj"
'Clear the screen to black on white
COLOR 0,7
CLS
'Dress up the screen a little with an instruction bar
'above and below our Dos window area.
COLOR 7,1
LOCATE 2,10: PRINT SPC(61)
LOCATE 2,28: PRINT "** Dos Shell In Progess **"
LOCATE 24,10: PRINT SPC(61);
LOCATE 24,25: PRINT "Type 'Exit' to return to Basic";
'Set up and call the DosWindow routine
TopRow% = 4:      LeftCol% = 8:           'Upper left corner
BottomRow% = 22:  RightCol% = 72          'Lower right
Colr% = 3:        LineWrap% = 0           'Color and wrap
Status% = DosWindow%(TopRow%, LeftCol%, BottomRow%,_
                     RightCol%, Colr%, LineWrap%)
IF Status% THEN
    PRINT "Bad coordinates were passed to DosWindow!"
    END
END IF
SHELL                       'Now, we're ready to shell!
'Don't forget this! Failure to call this routine before
'ending your program can result in a crash.
Call RemoveDosWindow
'Clear the screen to white on black and end.
COLOR 7,0
CLS
END
```

Polish Your Shell. This sample program generates screen output to test your image when SHELLing to a pre-defined window.

interrupt &H29 and interrupt &H10 during the SHELL. The interception of an interrupt service is referred to as "hooking an interrupt." While these interrupts are hooked, the DosWindow routine makes sure that all cursor positioning requests are kept within the bounds of a predefined window, that the desired color attribute is used to display data within the window, and that attempts to clear (CLS) or scroll the screen affect only the contents of the window (Listing 1). If an application requests an interrupt &H10 service that my routine isn't prepared to handle, the original interrupt &H10 service routine will handle it.

It's easy to use the DosWindow routine from QuickBasic, BasicPDS, VBDOS, or PowerBasic 3.0. Just add the following declarations to the top of the Basic program module that will be calling the DosWindow routines:

```
Declare Function DosWindow% (BYVAL _
        TopRow%, BYVAL LeftCol%, _
        BYVAL BottomRow%, BYVAL _
        RightCol%, BYVAL Colr%, _
        BYVAL LineWrap%)
Declare Function RemoveDosWindow ()
```

If you are using PowerBasic 3.0, you'll need one additional line:

```
$Link "DOSWIN.OBJ"
```

When declaring the routines, don't forget the BYVALs! If you leave them out, unpredictable results will occur!

The parameters TopRow%, LeftCol%, BottomRow%, and RightCol% define the upper-left and bottom-right corners of the window. Colr% defines the color attribute used when displaying data within the window. When Back% and Fore% represent a set of Basic background and foreground color values (as used with the COLOR statement), Colr% can be calculated as follows:

```
Colr%=(Back%OR(Fore%AND 16)\ 2)_
*16+(Fore%AND 15)
```

The LineWrap% parameter is used to determine if the DosWindow routine wraps or truncates a line that's wider than the defined window. Truncating generally provides a more readable display within a DosWindow. Pass a value of -1 to wrap text or 0 to truncate text.

Once the routines are declared, all you have to do is invoke the DosWindow% function immediately before your SHELL statement and call RemoveDosWindow immediately upon returning from the SHELL. Listing 2 contains a small example program called DWTEST.BAS.

Remember, the DosWindow routine can control the output of programs only when they use DOS or BIOS services to display data. Because Basic's PRINT statement uses direct video writes instead of BIOS and DOS services (direct writes are faster), DosWindow will not be able to control its output. There is a way around this: instead of using a regular PRINT statement in a program you want to SHELL, OPEN the CON (console) device and print through it. For example:

```
OPEN "CON" FOR OUTPUT AS #1
PRINT #1, "This text is routine _
through the BIOS and"
PRINT #1, "is trapable by the _
DosWindow routine!"
CLOSE #1
```

There you have it. Once you've integrated the DosWindow routine with your application, you can SHELL to the predefined window and maintain control of your user interface. To download the this code, call my BBS at 404-928-7111 and download a file called DOSWIN.EXE. It is a self-extracting archive that contains everything you need. n