Dr. Bob - *On Bugs, Reports & Fixes*

Do you know of any other bugs, reports or do you desperately need a fix? Please don't forget to ask for Dr.Bob's advice. And remember: ***If It Ain't Broke, Let Dr.Bob Have A Look At It.***

You can send your problems, bug reports or information on bug fixes to Dr. Bob either by email to **bobs@dragons.nest.nl**, to **[100434,2072]** on CompuServe, or by post to:

drs. Robert E. Swart
P.O. Box 799
5702 NP HELMOND
THE NETHERLANDS

The Pascal Magazine is a new magazine for Pascal developers, and includes: news, comment, product info & reviews (libraries/tools, books), "how to" articles for advanced & beginners, lists of libraries/tools, etc. Each issue will include a free disk of shareware libaries, code snippets, tools, and all code for that issue.

To register for one FREE complimentary copy, please send us your Name, Company (if any) and full postal address (including country). Please spread this around your friends and colleagues too!

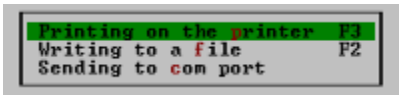
Chris Frizelle, Editor
The Pascal Magazine
41 Recreation Road Shortlands, Bromley Kent
BR2 0DY, United Kingdom
Tel/Fax: (+44) (0)81 460 0650
Email: 70630.717@compuserve.com

A Memory Slip in TPopupMenu

This report, from Hans van der Veeke in Baflo, The Netherlands, is about a minor bug in the Menus unit from the Turbo Vision Application Framework of Borland Pascal 7.0. The bug has been reported to Borland, but unfortunately has not been fixed in the maintenance release 7.01.
The problem concerns the objects `TMenuPopUp` and `TMenuBox`, both descendants of `TMenuView` from the Turbo Vision 2.0 unit Menus. These objects can be used to create stand-alone pop-up menus (in fact, the manual urges us to do so), for example in answer to a button-press we could pop-up a menu like this:

```
  R.Assign(25, 7, 50, 10);
  LocalMenu := New(PMenuBox, Init(R, NewMenu(
        NewItem('Printing on the ~p~rinter', 'F3', kbF1,    cmPrinter, hcPrinter,
        NewItem('Writing to a ~f~ile',       'F2', kbF2,    cmFile,    hcFile,
        NewItem('Sending to ~c~om port',     '',   kbNoKey, cmCom,     hcCom,
      nil)))),
    nil));
  Choice := DeskTop^.ExecView(LocalMenu);
  Dispose(LocalMenu, Done);
```

Execution of the code above gives the following pop-up menu:



This pop-up works just as good as any normal menu, including function keys and help-information (in the status line). The return value of the `ExecView` is the chosen item, counting up from 0.

Unfortunately, there is a bug in the Borland RTL responsible for the fact that not everything is cleared up all right. The inherited method `DisposeMenu` is not called, therefore the menu is not freed, and the (unusable) pointers remain allocated and are lost in cyberspace.

The simple fix is to create a descendant object from `TPopupMenu` with one inherited method, the destructor, as follows:

```
Unit GoodMenu;
interface
uses Menus;

Type
  PPopUpMenu = ^TPopUpMenu;
  TPopUpMenu = object (TMenuPopUp)
                 destructor Done; virtual;
               end {TPopUpMenu};

implementation

  destructor TPopUpMenu.Done;
  begin
    DisposeMenu(Menu);
    inherited Done
  end {TPopUpMenu};
end.
```

Now, whenever we place the GoodMenu unit within the uses-list of our program and define any pop-up menu as descendant from our `TPopupMenu`, then the menu will be correctly destroyed when the destructor is called. No more memory slip!

Unsupported DOS Interrupts from DPMI

This problem was brought to me by another friend, Wilbert van Leijen, from Amsterdam, The Netherlands. Wilbert had problems calling his usual country-specific upcase routine (using DOS interrupt $21 function $6520) from DPMI:

```
Function UpCase(C: Char): Char; Assembler;
ASM
      mov   DL,C
      mov   AX,$6520
    {$IFDEF DPMI}
      call  DOS3Call
    {$ELSE}
      int   $21
    {$ENDIF}
      mov   AL,DL
end {UpCase};
```

Even though we used the correct library macro `DOS3Call` in this case (it is not safe to do a straight DOS interrupt $21 under DPMI), the result of the function under DPMI is incorrect: no uppercasing is performed!
The reason for this is the fact that, although DOS interrupt $21, function $6520 is documented, the Borland DPMI Server does not support this call (as can be found in the *Borland Pascal Open Architecture Handbook*).
Fortunately, any self-respecting DPMI Server will allow us to send an interrupt to the "real" destination. We need DPMI interrupt $31 function $300, also called `SimulateRealModeInterrupt` to get the otherwise plain DOS interrupt $21, function $6520, where it should be processed: real mode DOS.

To make a long story short: the DPMI version of the country-specific UpCase function is as follows:

```
uses WinAPI;

Type
  ExtReg = record
             case Boolean of
               true: (LoLo,LoHi,HiLo,HiHi: Byte);
              false: (Lo,Hi: Word)
           end {ExtReg};

  TDPMIRegisters = record
                     EDI,ESI,EBP,Reserved,EBX,EDX,ECX,EAX: ExtReg;
                     Flags,ES,DS,FS,GS,IP,CS,SP,SS: Word
                   end {TDPMIRegisters};

  function SimulateRealModeInt(IntNo: Word;
                           var Regs: TDPMIRegisters): Word; Assembler;
  ASM
        mov   BX,IntNo { real mode interrupt nr in BL    }
        xor   CX,CX    { no copying bytes from the stack }
        les   DI,Regs  { ES:DI -> real TDPMIRegisters    }
        mov   AX,$300  { function $300 = real mode int   }
        int   $31      { call DPMI int $31 }
        jc    @Exit
        xor   AX,AX
   @Exit:
  end {SimulateRealModeInt};

  function UpCase(c: Char): Char;
  var DPMIRegs: TDPMIRegisters;
  begin
```

```
  FillChar(DPMIRegs,SizeOf(DPMIRegs),0);
  DPMIRegs.EDX.LoLo := Ord(c); { DL }
  DPMIRegs.EAX.Lo := $6520;
  SimulateRealModeInt($21,DPMIRegs); { <<< }
  UpCase := Char(DPMIRegs.EDX.LoLo)
end {UpCase};
```

Zombie-like Windows Delay

I'd like to report a bug on one of my own Tips & Tricks from the first issue of The Pascal Magazine. The Delay function for Windows from page 26 contains a rather serious bug. Dave Jewell was the first to report this bug to me, calling it a "classic bug" already.
The problem is that with the code as it stands, it's possible to accidentally "eat" `WM_QUIT` messages without noticing. This can be very bad news as it can cause an application to go into a "zombie" state where it sits in its message loop with all windows closed. The relevant bit of code should look something like this:

```
if PeekMessage(Msg, 0, 0, 0, PM_REMOVE) then
begin
  if Msg.message = WM_QUIT then
  begin
    { Ooops :- we've just eaten a WM_QUIT message    }
    { Pass it on to the owner window and abort delay }
    PostQuitMessage(Msg.wParam);
    Exit
  end;
  TranslateMessage(Msg);
  DispatchMessage(Msg)
end;
```

Afterword: *The African Chief (dr. A.A. Olowofoyeku, laa12@cc.keele.ac.uk) points out that using* `PostQuitMessage` *with the* `wm_Quit` *message will not work for WinCRT apps. It will result in something very strange indeed. With WinCRT apps, you need to use* `Halt(msg.WParam);` *- this gets processed instantly.*

Compiler Bug

A strange bug is the parser bug of the Turbo and Borland Pascal compiler, which generates an error when compiling the following type definition (brought to my attention by Kim K. from TurboPower Software):

```
Type
  x = record
        case system.boolean of
           true:(long:longint);
          false:(lo,hi:word);
      end;
```

The error is generated for the 'system.' part. If we just say "case boolean of" this code compiles without a problem. This is a strange glitch of the parser, since everywhere else we can use the 'system.' part before any identifier, routine or type from the system unit.

Another HeapEater in TDirView

Right after issue 1 was sent out, I received a message from Turbo Vision CyberTools author Steve Goldsmith (s.goldsmith2@genie.geis.com) about a Turbo Vision heap-eating bug in the object `TDirectoryViewer` (from `\BP\EXAMPLES\DOS\TVFM\DIRVIEW.PAS`).
To see the bug in action add a `THeapView` to `TVFM.PAS` (see file `TVDEMO\GADGETS.PAS`). This can be done in four steps: First, include the unit gadgets in the uses clause of TVFM (make sure gadgets.pas can be found by the compiler). Second, add a field Heap of type `PHeapView` to the `TMyApp` object. Third, put the following code in constructor `TMyApp.Init` (at the end):

```
GetExtent(R);
Dec(R.B.X);
R.A.X := R.B.X - 9;
R.A.Y := R.B.Y - 1;
Heap := New(PHeapView, Init(R));
Insert(Heap);
```

Fourth, be sure to call `Heap^.Update` in procedure `TMyApp.Idle`.

You can see the bug in action if you go to the root directory of a tree window and press '-' to contract all nodes. Press '*' to expand all nodes and note heap size. Now, pressing '*' again will cause the heap to decrease and not return.

The bug can be traced down to `procedure TDirectory.Adjust(Expand: Boolean);` It doesn't care if the tree is in an expanded state when allocating new nodes, so all the old nodes are left on the heap. To fix the problem we must decended a new object from `TDirectoryViewer` and add the following virtual method:

```
PDirViewer = ^TDirViewer;
TDirViewer = object(TDirectoryViewer)
               procedure Adjust(Node: pointer; Expand: Boolean); virtual;
             end {TDirViewer};

procedure TDirViewer.Adjust(Node: pointer; Expand: Boolean);
begin
  if Expand then
  begin
    if not IsExpanded(Node) then
      inherited Adjust(Node,Expand)
  end
  else
  begin
    if IsExpanded(Node) then
      inherited Adjust(Node,Expand)
  end
end {Adjust};
```

Finally, make sure to adjust the right call in `TREEWIN.PAS` (i.e. allocate a directory viewer of type `PDirViewer` instead of `PDirectoryViewer`).

With A Little Help From My HeapLimit

In the internet newsgroup comp.os.msdos.programmer.turbovision, Dr. John Stockton (jrs@merlin.dclf.npl.co.uk) posted the following problem, related to an advise given in The Pascal Magazine:

"As recommended in The Pascal Magazine Issue 1 p.12 by Ray Bernard, I have been testing a BP7.01 TV2.0 program with HeapLimit changed from the default 1024 to 0. I then found another range-like error (apart from the one in VALIDATE.PAS TFilterValidate), this time in TV2.0's HELPFILE.PAS."

John got an Error 216 - General Protection Fault - from the (only) line using "IsBlank" sometimes when a help window is expanded or drawn.
The offended line is:

```
  while (I>Offset) and not IsBlank(PCarray(@Text)^[I]) do Dec(I) ;
```

John himself found a bodge which prevents the Error 216. It appears that it is the FIRST (biggest) `"I"` only which is the problem; and the bodge is to alter `TParagraph` to contain an extra one byte after the beginning of Text.

Further examination by Dr. Bob showed the real kind of this bug: Since both `Offset` and `PCArray` start with 0 (zero-based), we cannot simply add the "Width" to the Offset, but we must add "Width-1" to the Offset.

This results in the following code:

```
{ TV 2.0: HELPFILE.PAS - line 471 and on, fix on line 475 }

begin
  I := Scan(Text, Offset, Size, #13);
  if (I >= Width) and Wrap then
  begin
    I := Offset + Width -1; {-1, since offset and PCArray starts with 0 }
    if I > Size then I := Size
    else
    begin
 {GPF} while (I > Offset) and not IsBlank(PCArray(@Text)^[I]) do Dec(I);
      if I = Offset then I := Offset + Width
      else Inc(I);
    end;
    if I = Offset then I := Offset + Width;
    Dec(I, Offset);
  end;
  TextToLine(Text, Offset, I, Line);
  if Line[Length(Line)] = #13 then Dec(Line[0]);
  Inc(Offset, I);
  WrapText := Line;
end;
```

I've checked this code, and it runs fine on all my helpfiles, even with HeapLimit set to 0. Just another thing to put out of our minds again.

How to Read a 'read-only' File

The African Chief, Dr. Abimbola Olowofoyeku (laa12@cc.keele.ac.uk) sent a fix on the internet for the problem that can be encountered when we try to open a read-only file. If we use the following code:

```
Assign(f,'filename.ext');
{$I-} reset(f,1); {$I+}
```

We get an error message and the file is not opened. The only way of 'fixing' this seemed to be to use the `DOS ATTRIB` command before or from inside the program to remove the 'read-only' attribute from the file. While this seems to work with local 'read-only' files, it will fail on files that cannot be made non-'read-only', like files from a restricted network drive or a CD-ROM drive!

The problem with reset is the fact that is opens the file not only for "read" access, but for "write" access as well. In fact, reset opens the file in "read/write" mode, which is specified in the local system variable FileMode. If we just give FileMode the value 0 before each reset, we only open the file for "read". A more in depth description of FileModes (and sharing files) will be found in an upcoming issue of The Pascal Magazine, but for the moment the following code will do:

```
var f: File;
    OldFileMode: Integer;
begin
  OldFileMode := FileMode; { save the original state of FileMode }
  FileMode := 0;           { read access only }
  Assign(f,'filename.ext');
  {$I-} Reset(f, 1);  {$I+}
  if IOResult <> 0 then { };
  { do your stuff here (and close the file afterwards) }
 FileMode := OldFileMode; {restore the original state of FileMode}
end;
```

Please not that FileMode does *not* work with Text files (the File Sharing article will deal with that too).

How to Make a TEdit "read-only"

How do I get an edit control Read-only? The ES_READONLY-attribute doesn't work. It works when I have a dialogbox with an edit control, but not when I create the control myself.

A thorough search by Dr. Bob revealed the edit message constant "EM_SETREADONLY" in Windows.H, and it turns out that sending this message to the TEdit control (even after you already made it with ES_READONLY) did the job:

```
Type
  TMyEdit = object (TEdit)
              procedure SetupWindow; virtual;
            end {TMyEdit};

  procedure TMyEdit.SetupWindow;
  begin
    TEdit.SetUpWindow;
    SendMessage(HWindow, em_SetReadOnly, Word(True), 0);
  end {SetupWindow};
```

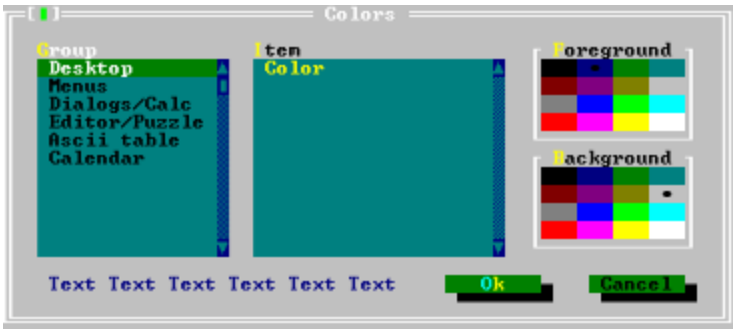
Background Colours in TColorSelector

Have you ever wondered why we can choose between 16 foreground colours, but only 8 background colours in Turbo Vision? It has to do with the fact that the 8 "bright" background colours are - by default - disabled on machines with a EGA/VGA card (or better). If we enable the bright colours, we lose the "blinking" attribute, hence the name "bright/blink switch".

We can to select the "bright" background colours using interrupt $10, function AX = $1003 (BL = 0 for bright, BL = 1 for Blink) as follows:

```
ASM
      mov   AX,$1003
      mov   BL,$00
      int   $10
end {BriteEGA};
```

Now, we also have to modify the Turbo Vision object `TColorSel`, because only room is made for the default eight background colours in the dialog. Our new "enhanced" version `TNewColorSel` enables us to select 16 foreground and 16 background colours, just like this:



The `unit NewColor` includes code for automatically selecting the "bright" background colours on entry, and selecting the default "blinking" colours on exit of the program (using a customised exitproc).

To get the above dialog, you have to make two modifications in TVDEMO.PAS:
- include the unit NewColor in the uses list (for example right after ColorSel)
- change every PColorDialog to PNewColorDialog (on line 431 and 433).

```
unit NewColor;
interface
uses Objects, Drivers, Views, Dialogs, ColorSel;

Type
  PNewColorDialog = ^TNewColorDialog;
  TNewColorDialog = object(TColorDialog)
                      constructor Init(APalette: TPalette; AGroups: PColorGroup);
                    end {TNewColorDialog};

implementation

Type
  PNewColorSelector = ^TNewColorSelector;
  TNewColorSelector = object(TColorSelector)
                        procedure HandleEvent(var Event: TEvent); virtual;
                      end {TNewColorSelector};


  procedure TNewColorSelector.HandleEvent(var Event: TEvent);
  const MaxCol = $F;
        Width  =  4;
  var Mouse: TPoint;
      OldColor: Byte;

      procedure ColorChanged;
      var Msg: Integer;
      begin
        if SelType = csForeground then Msg := cmColorForegroundChanged
                                  else Msg := cmColorBackgroundChanged;
        Message(Owner, evBroadcast, Msg, Pointer(Color))
      end {ColorChanged};

  begin
    case Event.What of
      evMouseDown:
        begin
          OldColor := Color;
          repeat
            if MouseInView(Event.Where) then
            begin
              MakeLocal(Event.Where, Mouse);
              Color := Mouse.Y * 4 + Mouse.X div 3
            end
            else Color := OldColor;
            ColorChanged;
            DrawView
          until not MouseEvent(Event, evMouseMove)
        end;
      evKeyDown:
        begin
          case CtrlToArrow(Event.KeyCode) of
            kbLeft:
              if Color > 0 then Dec(Color)
                           else Color := MaxCol;
            kbRight:
              if Color < MaxCol then Inc(Color)
                                else Color := 0;
            kbUp:
              if Color > Width - 1 then Dec(Color, Width)
              else
                if Color = 0 then Color := MaxCol
```

```
                        else Inc(Color, MaxCol - Width);
          kbDown:
            if Color < MaxCol - (Width - 1) then Inc(Color, Width)
            else
              if Color = MaxCol then Color := 0
                                else Dec(Color, MaxCol - Width);
          else
          begin
            inherited HandleEvent(Event);
            Exit
          end
        end
      end;
    evBroadcast:
      if Event.Command = cmColorSet then
      begin
        if SelType = csBackground then Color := Event.InfoByte SHR Width
                                  else Color := Event.InfoByte AND MaxCol;
        DrawView;
        Exit
      end
      else
      begin
        inherited HandleEvent(Event);
        Exit
      end
    else
    begin
      inherited HandleEvent(Event);
      Exit
    end
  end;
  DrawView;
  ColorChanged;
  ClearEvent(Event)
end {HandleEvent};


constructor TNewColorDialog.Init(APalette: TPalette; AGroups: PColorGroup);
var R: TRect;
    P: PView;
begin
  R.Assign(0, 0, 61, 18);
  TDialog.Init(R, 'Colors');
  Options := Options or ofCentered;
  Pal := APalette;

  R.Assign(18, 3, 19, 14);
  P := New(PScrollBar, Init(R));
  Insert(P);
  R.Assign(3, 3, 18, 14);
  Groups := New(PColorGroupList, Init(R, PScrollBar(P), AGroups));
  Insert(Groups);
  R.Assign(2, 2, 8, 3);
  Insert(New(PLabel, Init(R, '~G~roup', Groups)));

  R.Assign(41, 3, 42, 14);
  P := New(PScrollBar, Init(R));
  Insert(P);
  R.Assign(21, 3, 41, 14);
  P := New(PColorItemList, Init(R, PScrollBar(P), AGroups^.Items));
  Insert(P);
  R.Assign(20, 2, 25, 3);
```

```
    Insert(New(PLabel, Init(R, '~I~tem', P)));

    R.Assign(45, 3, 57, 7);
    ForSel := New(PColorSelector, Init(R, csForeground));
    Insert(ForSel);
    Dec(R.A.Y); R.B.Y := R.A.Y+1;
    ForLabel := New(PLabel, Init(R, '~F~oreground', ForSel));
    Insert(ForLabel);

    Inc(R.A.Y, 7); Inc(R.B.Y,10);
    BakSel := New(PNewColorSelector, Init(R, csBackground));
    Insert(BakSel);
    Dec(R.A.Y); R.B.Y := R.A.Y+1;
    BakLabel := New(PLabel, Init(R, '~B~ackground', BakSel));
    Insert(BakLabel);

    R.Assign(3, 15, 34, 16);
    Display := New(PColorDisplay, Init(R, NewStr(' Text')));
    Insert(Display);

    R.Assign(44, 3, 59, 8);
    MonoSel := New(PMonoSelector, Init(R));
    MonoSel^.Hide;
    Insert(MonoSel);
    R.Assign(43, 2, 49, 3);
    MonoLabel := New(PLabel, Init(R, '~C~olor', MonoSel));
    MonoLabel^.Hide;
    Insert(MonoLabel);

    if (AGroups <> nil) and (AGroups^.Items <> nil) then
      Display^.SetColor(Byte(Pal[AGroups^.Items^.Index]));

    R.Assign(36, 15, 46, 17);
    P := New(PButton, Init(R, 'O~k~', cmOk, bfDefault));
    Insert(P);
    R.Assign(48, 15, 58, 17);
    P := New(PButton, Init(R, 'Cancel', cmCancel, bfNormal));
    Insert(P);
    SelectNext(False)
  end {Init};


var SaveExit: pointer;

procedure NewExit; far;
begin
  ExitProc := SaveExit;
  ASM
        mov   AX,$1003
        mov   BL,$01
        int   $10
  end {BlinkEGA};
end {RunExit};


begin
  SaveExit := ExitProc;
  ExitProc := @NewExit;
  ASM
        mov   AX,$1003
        mov   BL,$00
        int   $10
  end {BriteEGA};
```

```
end.
```