The Delphi CLINIC

Edited by Bob Swart

Bring your problems to our panel of experts!

If there's something puzzling you about an aspect of Delphi, just email the Delphi Clinic Editor, Bob Swart, at Compuserve 100434,2072 or write or fax us at The Delphi Magazine

Q Well I've seen some strange antics in library code in my time, but this one takes the cookie. Please explain the programming methodology behind this in file CONTROLS.PAS:

```
procedure Hack;
var
 W: array[0..$7F] of Byte;
 I: Integer;
begin
  for I := 0 to $7F do
    W[I] := I;
end;
```

called only once in the same file just below the declaration of `Hack` as shown in Listing 1.

A That loop code is required to activate a poorly documented and understood feature of Windows: that Windows standard controls will paint into a DC that you provide, instead of doing their own `BeginPaint/EndPaint` cycles. However, for at least one standard Windows control, a zone of downstream stack must be initialised in a certain way before the trick will work.

This trick enables us to print entire forms to a printer DC, including buttons and such that do their own painting. This technique is used by several Microsoft products, so we can feel reasonably safe in using it ourselves...

[Answer by Danny Thorpe]

Q My menu shortcut keys don't seem to work correctly, can you help?

A There appears to be a bug in the VCL source file MENUS.PAS. Listing 2 shows the original procedure `ShortCutToKey` from the `Menus` unit and also how it should look.

[Answer by Michael Williams]

Q How do I navigate through my numerous forms?

A Apart from using the Form list, there is another way. If you've got a form open, pressing the F12 function key puts you in the code editor with that form's code loaded. Pressing F12 in the code editor then takes you to the form corresponding to that code. So the editor can serve as a navigation aid for getting bewteen forms. I use it that way a lot!

[Answer by Neil Rubenking]

Q When I'm building a DLL, why does Delphi insist on having a uses clause in my source file, even though I don't want to 'use' any units?

A If your DLL source file does not have a uses clause somewhere (in either the main part in case of a program or library, or in the interface or implementation section of a unit), Delphi will give you an error message: `Error in module X: USES clause is missing or incorrect`. However, if you just click OK, you can continue to work with, edit and compile this 'bad' file with no problem. I guess the 'error' should be made a 'warning' (or be removed completely).

➤ *Listing 1*

```
procedure TWinControl.DefaultHandler(var Message);
begin
  if FHandle <> 0 then with TMessage(Message) do begin
    if (Msg = WM_PAINT) and (WParam <> 0) then Hack;
    Result := CallWindowProc(FDefWndProc, FHandle, Msg, WParam, LParam);
  end else
    inherited DefaultHandler(Message);
end;
```

➤ *Listing 2*

```
{** ORIGINAL VERSION: }
procedure ShortCutToKey(ShortCut: TShortCut; var Key: Word;
  var Shift: TShiftState);
begin
  Key := Key and not (scShift + scCtrl + scAlt);
  Shift := [];
  if Key and scShift <> 0 then Include(Shift, ssShift);
  if Key and scCtrl <> 0 then Include(Shift, ssCtrl);
  if Key and scAlt <> 0 then Include(Shift, ssAlt);
end;

{** THIS IS HOW IT SHOULD READ: }
procedure ShortCutToKey(ShortCut: TShortCut; var Key: Word;
  var Shift: TShiftState);
begin
  Key := ShortCut and not (scShift + scCtrl + scAlt);
  Shift := [];
  if ShortCut and scShift <> 0 then Include(Shift, ssShift);
  if ShortCut and scCtrl <> 0 then Include(Shift, ssCtrl);
  if ShortCut and scAlt <> 0 then Include(Shift, ssAlt);
end;
```

Q Why can't I debug my ObjectPascal DLL? The integrated debugger is unable to debug it, while Turbo Debugger 3.2 (from Borland Pascal) reports an incorrect linker version.

A First of all, you do need the stand-alone debugger in order to debug DLLs. Secondly, you need the new Turbo Debugger 4.6, which ships as part of the Delphi RAD Pack, or with Turbo Assembler 4.0, or in the Borland C++ 4.51 update. Personally, I'd go for the RAD Pack!

Q How do I make my Delphi application's window stay on top of the other windows which are open?

A Look at the `fsStayOnTop` setting of the `TForm` property `FormStyle`. This will cause the window to always stay on top of whatever other windows are open. However, when you minimize the window to an icon, it no longer stays on top like, for example, the Clock application that ships with Windows itself. Since an icon is not a form, this does not sound *that* strange, but if anyone has a clue how to fix this, we would be very interested in hearing about it!

Q I would like help on image handling in Delphi. I currently use the ImageKnife VBX and it works fine(ish), but I would prefer to do it all without a VBX. The sort of functionality I need is to load and save bitmaps, show them on the screen and get direct pixel access to the actual pixel data. The Delphi `TImage` component does most of this, but how do I directly access the image data?

A If you have a `TImage` component, you can get to its pixel data by using the `Pixels` property of the bitmap's `Canvas`, eg if you have a component `Image1` on a form with a picture in it, say the file C:\IMAGES\MYPIC.BMP, and a button called `Button1` you can use the code shown in Listing 3.

Also, the equivalent of reading and writing `Pixels[x,y]` is:

```
dc := (image1.picture.graphic
  as tbitmap).canvas.handle;
col := GetPixel(dc, x, y);
SetPixel(dc, x, y, col);
```

[Answer by Brian Long]

Whoops...

And now for the apologies! Thanks to Brian Long for pointing out a couple of goofs in Issue 2's Delphi Clinic column.

First, a reader wanted his application to first of all show a login dialog and then, if the login was not accepted, to close the application *without* the app's main form being displayed. The answer contained the elements of what was needed, but wasn't too well explained. The code to handle this situation needs to go in the `FormCreate` method for the main form, which handles the `OnCreate` event for the form. The reader had found that testing the result of the dialog and then using `Close` did not have the desired effect. The key is to use a call to `Application.Terminate` instead:

```
procedure TFormMain.FormCreate(
  ...)
var
  LoginForm : TUserLogin;
begin
  Application.CreateForm(
    TUserLogin, LoginForm);
  if LoginForm.ShowModal =
    mrCancel then begin
    LoginForm.Free;
    Application.Terminate;
  end;
end {FormCreate};
```

This should do the trick: the call to `Terminate` will kill the program before the main form is displayed.

Secondly, a reader queried how to "shell" out from a Delphi application and call another executable and then return to the Delphi application once the user had exited the "shelled" program. The code given in Listing 3 (from Issue 2) was incorrect. Brian suggests the code in Listing 4 (this issue!) as a replacement.

[Editor's note to Dr.Bob's boss – Hey, this guy needs a holiday, you're working him too hard!]

➤ *Listing 3*

```
procedure TForm1.Button1Click(Sender: TObject);
var LoopX, LoopY: Word;
begin
  { save typing this all the time }
  with Image1.Picture do
    { Canvas property is only in bitmaps }
    if Graphic is TBitmap then
      with Graphic as TBitmap do
        { invert all pixels a column at a time }
        for LoopX := 0 to Pred(Width) do begin
          for LoopY := 0 to Pred(Height) do begin
            Canvas.Pixels[LoopX, LoopY] :=
              clWhite - Canvas.Pixels[LoopX, LoopY];
          end;
          { standard technique to stop flickering on VGA }
          while Port[$3DA] and 8 = 0 do;
          { be multi-user friendly - yield after each column }
          Application.ProcessMessages;
          { also let user terminate app }
          if Application.Terminated then Break;
        end;
end;
```

➤ *Listing 4*

```
var InstID: THandle;
begin
  InstID := WinEXec(CmdLine, CmdShow);
  Result := InstID >= hInstance_Error;
  if Result then
    repeat
      Application.ProcesssMessages;
    until (GetModuleUsage(InstID) = 0) or Application.Terminated;
end;
```