The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?

Just email Brian Long, our Delphi Clinic Editor, on clinic@blong.com or write/fax us at The Delphi Magazine

Making A 256 Colour Bitmap

Q I am writing a part of my application where I need to capture a (known) area of the screen and save it to a .BMP file. This would be fine, I can write the API code to do this, but the screen is in 256 colour. When I check the bitmap file it only has 16 colours stored in it. What do I need to do in addition to just saving the bitmap to get all the right colours saved into the file?

A What you are missing is a colour palette. Just creating a `TBitmap` object, writing onto its canvas and calling the `SaveToFile` method will make you a 16 colour bitmap. To get a 256 colour bitmap you need to manufacture a palette and assign it to the bitmap's `Palette` property. Fortunately, Windows already has a palette set up internally that the other applications are making use of. Since you are capturing the screen then this will be the palette you want.

On the other hand, if you were programmatically manufacturing an arbitrary bitmap, then you might need to manufacture a custom palette with custom colour values in it.

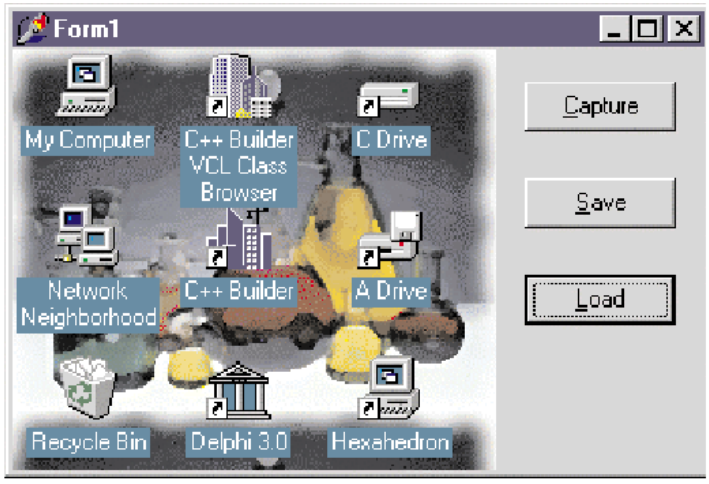
In Delphi, a palette is represented by a `TLogPalette` (see Listing 1). This is a data structure that can hold a variable number of `TPaletteEntry` records. Since the actual number to be used may vary, the array has been declared with just one element.

To use a `TLogPalette` structure, you typically declare a variable that points to it (ie a `PLogPalette`). You can then allocate sufficient heap space for the fixed fields and also for as many palette entries as are required.

Delphi 3 has rather more palette support built into the VCL and so to avoid doing repetitive heap management calls, it defines a new type that specifies 256 palette entries (see Listing 2).

Your code that fails to save the palette probably looks a bit like Listing 3. It should be extended to look more like Listing 4 which is one of the button `OnClick` handlers from the Capture.Dpr project on the disk. This has an image control called `Img` and three buttons on it. The first button captures the top left portion of the screen (as big an area as the image started its life with). The other two buttons save the image to a BMP file and load a BMP file into the image (to verify

➤ *Figure 1*



➤ *Listing 1*

```
PPaletteEntry = ^TPaletteEntry;
TPaletteEntry = packed record
  peRed: Byte;
  peGreen: Byte;
  peBlue: Byte;
  peFlags: Byte;
end;
PLogPalette = ^TLogPalette;
TLogPalette = packed record
  palVersion: Word;
  palNumEntries: Word;
  palPalEntry: array[0..0] of TPaletteEntry;
end;
```

➤ *Listing 2*

```
PMaxLogPalette = ^TMaxLogPalette;
TMaxLogPalette = packed record
  palVersion: Word;
  palNumEntries: Word;
  palPalEntry: array [Byte] of TPaletteEntry;
end;
```

that the palette was saved correctly). Figure 1 shows the program after having captured a corner of my 256 colour desktop, saved it and reloaded it from a BMP file.

To accompany the Capture.Dpr project, a Capture3.Dpr project is also supplied. This uses Delphi 3's new picture save and picture load dialogs and also uses `TMaxLogPalette` instead of `PLogPalette` to avoid memory management operations.

Background Palette Information

When you are running Windows in 16 colour mode, there are 4 bits required to represent the total number of colours and so 16 colour mode is also referred to as 4-bit mode. Each pixel in the bitmap takes 4 bits to indicate which of the 16 colours it is. These 16 colours are used by the video system to index into the full range of colours available. 256 colour mode uses 8 bits to represent each of the 256 colours and again indexes into a colour table in the video system. Therefore 256 colour mode is often referred to as 8-bit mode. A palette can be used to specify alternative target colours for the index values to refer to in both 4-bit and 8-bit mode.

If you are working in high colour modes such as 16-bit or 24-bit, then the issue of palettes becomes somewhat redundant. In 16-bit colour mode (not to be confused with 16 colour mode, which is 4-bit), each pixel uses 16 bits to represent the colour. Rather than an index, this value is made up of three sets of five bits (and a spare) that define the colour in terms of how much red, green and blue go to make it up. In 24-bit, there are three sets of eight bits to define colours even more accurately. So palettes to define the target colours are mostly irrelevant: the pixel values themselves define the colours.

MessageDlg Surprises

Q I regularly use `MessageDlg` with `OK` and `Cancel` buttons. Recently I used one with a `Yes` and a `No` button and got a nasty shock. If the user presses neither button but closes the dialog with the dialog's system menu, then neither `mrYes` nor `mrNo` is returned by `MessageDlg`. What should I do to trap for this possibility?

A I guess that you are using code like that in Listing 5. It is standard Windows practice to take a user-action of closing the form with `Alt-F4` or whatever (ie no button pressing) to be a cancellation, so the `MessageDlg` routine returns `mrCancel` under those circumstances. Therefore you could modify your code as shown in Listing 6. Alternatively you could embrace that third option and differentiate between `No` and `Cancel`. `No` answers "No" to the

➤ *Listing 3*

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R: TRect;
  C: TCanvas;
begin
  R := Img.BoundsRect;
  C := TCanvas.Create;
  C.Handle := GetDC(HWnd_Desktop);
  try
    Img.Canvas.CopyRect(R, C, R);
  finally
    ReleaseDC(HWnd_Desktop, C.Handle);
    C.Free;
  end;
end;
```

➤ *Listing 4*

```
procedure TForm1.Button1Click(Sender: TObject);
const
  NumColors = 256;
var
  R: TRect;
  C: TCanvas;
  LP: PLogPalette;
  Size: Integer;
begin
  R := Img.BoundsRect;
  C := TCanvas.Create;
  C.Handle := GetDC(HWnd_Desktop);
  try
    Img.Canvas.CopyRect(R, C, R);
    Size := SizeOf(TLogPalette) +
     (Pred(NumColors) * SizeOf(TPaletteEntry));
    LP := AllocMem(Size);
    try
      LP^.palVersion := $300;
      LP^.palNumEntries := NumColors;
      GetSystemPaletteEntries(C.Handle, 0, NumColors, LP^.palPalEntry);
      Img.Picture.Bitmap.Palette := CreatePalette(LP^);
    finally
      FreeMem(LP, Size);
    end
  finally
    ReleaseDC(HWnd_Desktop, C.Handle);
    C.Free;
  end;
end;
```

➤ *Listing 5*

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Save changes before leaving?',
       mtInformation, [mbYes, mbNo], 0) = mrNo then
    Exit;
  { Do saving ... }
end;
```

question being asked but allows the execution flow to continue unmolested, whereas Cancel stops the current execution flow. See Listing 7 for an example of this.

Localised Database Headaches

Q I have a problem concerning Personal Oracle 7.3 on Windows 95 and using BDE 3.5. If I try to write data into an integer field in my database and have a German version of Personal Oracle installed I get an exception saying that the number format is not valid. Dates and strings work without problem. Also the English version of Personal Oracle does not exhibit this problem. I tried all kinds of different BDE settings to no avail. What must I do to fix the problem?

A This is quite a well known problem for anybody who uses a comma as a decimal separator as opposed to a period (or full stop). The BDE Configuration app's Number page has a likely looking setting called DECIMALSEPARATOR, but unfortunately it is ignored by Delphi. In fact all the settings on the last three pages are ignored by Delphi: they are principally meant to be used for QBE execution and since Delphi does not inherently support QBE files, the settings are not used.

The problem occurs primarily when you set the ENABLE BCD property to True when working with Oracle and Informix. This is a 32-bit BDE setting which defaults to False.

Luckily there are two relatively painless workarounds. One would be to set ENABLE BCD back to False. The other involves issuing this query:

```
ALTER SESSION
  SET NLS_NUMERIC_CHARACTERS = '.,'
```

after the database has been opened but before you open any of the datasets. You need to ensure the alias's SQLPASSTHRU MODE is set to either SHARED AUTOCOMMIT or SHARED NOAUTOCOMMIT so that the datasets will actually benefit from this query statement.

Iconic Applications

Q I am trying to start up an application with it always minimised on the task bar (Win95 or NT 4) or minimised on the desktop (WinNT 3.51 or Win 3.1x). I am unable to come up with a satisfactory solution.

A In theory this is easy. When someone tries to restore your iconised form, Windows sends it a message. If the form returns zero, the form doesn't restore. That should be it. However, things are often not that clear cut when talking to Delphi forms at the API level.

To make the application start off minimised, you set the main form's WindowState property to wsMinimized. This causes a Delphi 1 app to start life with the main form iconised on the task bar or desktop. In the case of a Delphi 3 app, it is the Application window that starts off minimised on the desktop. That is more like what would be anticipated by someone who knows about the innards of the VCL since when any Delphi application is running, minimising the main form causes the Application window to be iconised on the task bar, not the main form's window. When a Delphi 2 app starts with its main form set to minimise it gets things confused and leaves the main form looking like a minimised MDI child above the task bar. The fix for this was discussed in *The Delphi Clinic* in Issue 9. When the fix is applied, Delphi 2 and 3 act alike.

So, a Delphi 1 app needs to prevent the main form being restored and Delphi 2 and 3 apps need to prevent the Application window being restored. Of course what normally happens when the Application window is restored is that it restores all the other windows (including the minimised main form) so 32-bit apps still really need to prevent the main form being restored. This business with the main form is easily dealt with using a message handling method to respond to wm_QueryOpen messages.

To make the application's system menu look sensible, during application startup we can delete the inappropriate menu items such as Move, Size, Restore, Minimize and Maximize, leaving the only relevant one: Close. Of course in Delphi 1 this is the main form's system menu.

In Delphi 2 and 3 it will be the Application's. This can also be done in a message handler. Windows sends a wm_NCCreate message when it is manufacturing the non-client area of the form (including borders and system menu). We can trap this message and follow this default functionality with the appropriate API calls to tidy up the system menu.

➤ *Listing 6*

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if MessageDlg('Save changes before leaving?',
       mtInformation, [mbYes, mbNo], 0) <> mrYes then
    Exit;
  { Do saving ... }
end;
```

➤ *Listing 7*

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  case MessageDlg('Save changes before leaving?',
       mtInformation, mbYesNoCancel, 0) of
    { Save data if user wishes }
    mrYes: { Do saving ... };
    { Abandon form closure }
    mrCancel: CanClose := False;
  end;
end;
```

The IconStay.Dpr project on the disk implements all these requirements using conditional compilation to work in all three versions of the product.

Listing 8 shows the main form unit, although it omits the code required to fix the Delphi 3 bug.

Acknowledgements

Thanks to Steve Axtell and Mike Scott for help with parts of this month's column.

➤ *Listing 8*

```
type
  TForm1 = class(TForm)
  private
    procedure WMQueryOpen(var Msg: TWMQueryOpen);
      message wm_QueryOpen;
    procedure WMNCCreate(var Msg: TWMNCCreate);
      message wm_NCCreate;
  end;
...
procedure TForm1.WMQueryOpen(var Msg: TWMQueryOpen);
begin
  Msg.Result := 0;
end;
procedure TForm1.WMNCCreate(var Msg: TWMNCCreate);
var SysMenu: HMenu;
begin
  inherited;
  { Delphi 1 generally shows the Application window
    when iconised. However if the app starts with the
    main form minimised then the icon is that of the
    main form. It seems that Delphi 2 and 3+ make
    things more consistent by using the Application
    window regardless. Therefore, some conditional
    compilation is required }
  SysMenu := GetSystemMenu(
    {$ifndef Windows}
    Application.{$endif}Handle, False);
  { Get rid of irrelevant system menu entries }
  DeleteMenu(SysMenu, sc_Size,  mf_ByCommand);
  DeleteMenu(SysMenu, sc_Move,  mf_ByCommand);
  DeleteMenu(SysMenu, sc_Minimize,  mf_ByCommand);
  DeleteMenu(SysMenu, sc_Maximize, mf_ByCommand);
  DeleteMenu(SysMenu, sc_Restore,  mf_ByCommand);
  { Get rid of the separator item that remains }
  DeleteMenu(SysMenu, 0, mf_ByPosition);
  { Refresh the system menu }
  DrawMenuBar(Handle);
end;
```