Getting Out Of a Dither

by Karl E. Peterson and Phil Weber

Click & Retrieve Source CODE!

Q **256-COLOR OUTPUT SPOILED**

Although I can get custom palettes in 256-color mode fine by setting a form'sPaletteMode property to vbPaletteModeUseZOrder, the vbPaletteModeCustom palette mode seems to be broken. When I try to use colors I know are in the custom palette, VB still dithers my output.

A VB5 introduced some interesting new ways to work with palettes. But even so, Windows maintains its intrinsic behavior. The problem is that by default, in 256-color mode, Windows' Graphic Device Interface (GDI) dithers all colors with a pattern composed of the eight main system colors (see Figure 1 and Listing 1). Work around the dithering by passing color references that have been altered to have special meaning to GDI. Two options provide the true, nondithered rendering of the desired colors.

Typically, a red-green-blue (RGB) color reference uses only three of the four available bytes to specify the red, green, and blue components of a requested color. Define a palette-relative RGB value as a normal RGB value, but set the high-order byte to 2. This usage causes Windows to return the palette entry that most closely matches the requested color and that is always a pure, nondithered color. To specify such a value, simply Or your RGB value with &H2000000:

```
Me.BackColor = &H2000000 Or RGB(255, 133, 0)
```

Another option is to use a specific palette index. Setting the high-order byte to 1 and the low-order byte to the desired index causes Windows to return a pure, nondithered color representing that specific palette entry. This value—represented in hex—would appear as &H10000nn, where nn is the desired palette index.

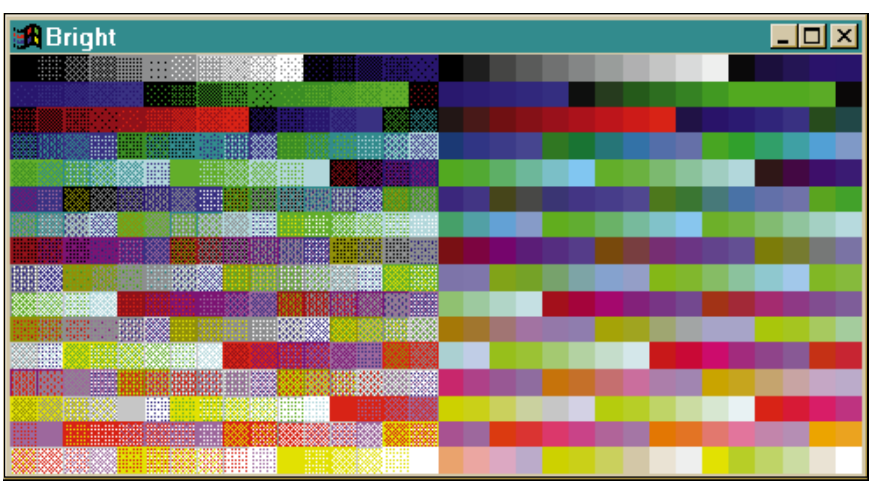

FIGURE 1 ***What a Difference Not Dithering Makes!*** *Windows dithers the color patches on the left using a pattern created from the eight primary system colors. On the right is an example of how richly colored native Visual Basic graphics can be when you avoid dithering. You can create this output by using the code shown in Listing 1.*

Karl E. Peterson is a GIS analyst with a regional transportation planning agency and serves as a member of the Visual Basic Programmer's Journal *Technical Review and Editorial Advisory Boards. Based in Vancouver, Washington, he's also an independent programming consultant specializing in ActiveX controls. In addition to contributing to various journals, Karl coauthored* Visual Basic 4 How-To*, from Waite Group Press. Online, he's a Microsoft MVP, and a section leader in both* VBPJ *online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.*

Phil Weber is an independent consultant specializing in Visual Basic and Web site development. He is a Microsoft Certified Solution Developer and Product Specialist. Find more of Phil's VB tips on his Web site: http://www.teleport.com/~pweber.

Ask the VB Pro provides you with free advice on programming obstacles, techniques, and ideas. Read more answers from our crack VB pros on the Web at http://www.inquiry.com/thevbpro. You can submit your questions, tips, and ideas on the site, or access a comprehensive database of previously answered questions.

VB5

```
Private Declare Function GetPaletteEntries Lib "gdi32" _
  (ByVal hPalette As Long, ByVal wStartIndex As Long, _
  ByVal wNumEntries As Long, lpPaletteEntries As _
  PALETTEENTRY) As Long

Private Type PALETTEENTRY
  peRed As Byte
  peGreen As Byte
  peBlue As Byte
  peFlags As Byte
End Type

Private m_pe(0 To 255) As PALETTEENTRY

Private Sub Form_Click()
  Me.Refresh
End Sub

Private Sub Form_Load()
  With Me
     .PaletteMode = vbPaletteModeCustom
     Set .Palette = LoadPicture("c:\vb5\bright.dib")
     Call GetPaletteEntries(.Palette.hPal, 0, 256, m_pe(0))
  End With
End Sub

Private Sub Form_Paint()
  Dim x As Long, y As Long
  Dim clr As Long
  Me.Scale (0, 0)-(32, 16)
  For y = 0 To 15
     For x = 0 To 15
        With m_pe(y * 16 + x)
           clr = RGB(.peRed, .peGreen, .peBlue)
        End With
        Me.Line (x, y)-(x + 1, y + 1), clr, BF
        Me.Line (x + 16, y)-(x + 17, y + 1), _
           &H2000000 Or clr, BF
     Next x
  Next y
End Sub
```

LISTING 1 ***Demand Nondithered Colors.*** *By default, Windows likes to dither nonstandard colors. Setting the high-order byte of a color reference to 2 lets you override this behavior. This method of color specification works in all versions of VB and Windows, and is extremely useful when calling GDI functions directly. Because the Palette and PaletteMode properties were introduced in VB5, prior versions require establishing palettes directly through the API. This code produces the output shown in Figure 1.*

Q WHERE'S THE OBJECT?

Given an object reference, how can I determine whether the object—an ActiveX EXE, for example—is running remotely or locally? If it's running remotely, how can I find out what machine it's on?

A If you know either the ProgID or CLSID of the object, you can get this information using an obscure method. The VB5 help file alludes to the solution, but leaves out one critical piece of information—that the first thing you must do is set a project reference to the Remote Automation Registry library, RacReg (racreg32.dll).

RacReg exposes RegClass, a class that offers a handful of methods used to read and write registry entries related to remote ActiveX components and control Remote Automation. Create a new instance of the RegClass class, and pass either the ProgID or CLSID of your object to the RegClass.GetAutoServerSettings method (see Listing 2).

If your component is running remotely, GetAutoServerSettings returns a Variant that contains an array of values about the component. Test the first element for True or False to determine whether the component is registered remotely. If it is, the second element reveals the name of the server where the component is registered. The remaining two elements tell you the RPC protocol and authentication level in use.

VB4 32-bit VB5

```
Private Sub CheckServerLocation()
  Dim oRegClass As RegClass
  Dim vRC As Variant

  Set oRegClass = New RegClass
  vRC = _
     oRegClass.GetAutoServerSettings("kpINI.CIniFile")
  If Not (IsEmpty(vRC)) Then
  If vRC(1) Then
        MsgBox "kpINI is registered remotely on a server named: " _
           & vRC(2)
     Else
        MsgBox "kpINI is registered locally."
     End If
  End If
End Sub
```

LISTING 2 ***Determine Whether a Component is Local or Remote.*** *This code, with a minor modification to correct an error in the help file, shows how to obtain the name of the server that an ActiveX component is registered on. Note that to run this, you must first establish a project reference to the RacReg library.*

Q RETURNING MOUSE COORDINATES

Many of the controls I write with VB5 are owner-drawn. That is, rather than put other controls within my UserControl, I fully create my own control using a slew of API calls to do all the drawing. I prefer to use Pixels as the ScaleMode for the UserControl, but I'd like mouse coordinates always to be returned in the appropriate scale—generally Twips, but it could be any scale. How can I raise a MouseMove event using the same coordinate system as my control's container?

A The UserControl object exposes two methods made to order for this problem—ScaleX and ScaleY. These methods convert a specified number of units from one unit of measure to another. In the UserControl's MouseMove event, use these methods to convert Pixels into whatever units the control's owner is using:

```
Private Sub UserControl_MouseMove _
   (Button As Integer, _
   Shift As Integer, x As Single, _
   y As Single)
   RaiseEvent MouseMove(Button, _
```

```
      Shift, ScaleX(x, vbPixels, _
      Extender.Parent.ScaleMode), _
      ScaleY(y, vbPixels, _
      Extender.Parent.ScaleMode))
End Sub
```

Note the use here of the Extender object. To be on the safe side, it's always wise to incorporate error trapping when using this object, as each potential control host is likely to expose a different set of properties for this object.

Q CONNECTING TO THE INTERNET

How can I make my app connect to the Internet automatically, the way Internet Explorer does when you enter a URL and you're not online?

A

Apps running under 32-bit versions of Windows can use the InternetAutodial and InternetAutodialHangup functions in wininet.dll:

```
Private Const _
   INTERNET_AUTODIAL_FORCE_UNATTENDED _
   As Long = 2

Private Declare Function _
   InternetAutodial Lib _
   "wininet.dll" (ByVal dwFlags _
   As Long, ByVal dwReserved _
   As Long) As Long

Private Declare Function _
   InternetAutodialHangup Lib _
   "wininet.dll" _
   (ByVal dwReserved As Long) _
   As Long

' To connect (returns non-zero
' if successful):

lResult = InternetAutodial( _
   INTERNET_AUTODIAL_FORCE_ _
   UNATTENDED, 0&)

' To hang up
lResult = InternetAutodialHangup(0&)
```

Early releases of Windows 95 did not include wininet.dll; it first shipped with Internet Explorer 3.0. If you use it in an application that you intend to distribute to others, you should include wininet.dll in your setup. To do so legally, you must use one of the redistributable files Microsoft provides for this purpose. VB5 includes such a file (axdist.exe) in the \VB\SetupKit\Kitfil32\Sys32 directory on the VB5 CD. VB4 users can download the redistributable IE3 files from the ActiveX SDK, at *http://support.microsoft.com/download/support/mslfiles/axredist.exe*. ×

Code Online

You can find all the code published in this issue of VBPJ on The Development Exchange (DevX) at http://www.windx.com. For details, please see "Get Extra Code in DevX's Premier Club" in Letters to the Editor.

Getting Out of a Dither

Locator+ Codes

Listings for the entire issue, plus a project file demonstrating use of custom palettes in VB5 and the main form used in the palette demonstration (free Registered Level): VBPJ0498
Listings for this article only, plus the files described above (subscriber Premier Level): AP0498