Print Text at An Angle

by Karl E. Peterson

Click & Retrieve Source CODE!

Q Rotating Text

My application graphs its own data, instead of relying on yet another control for this task. All's well except that I'd like to be able to print some text—for example, the y-axis title—at an angle. I can't find any VB methods to support this, and am assuming I'll need to turn to the API.

A Right you are. You need to create a logical font, specifying all the characteristics you want. Once you create the font, you can select it as the current font into any device context. I assume you're using either a PictureBox control or the form itself to draw upon. If so, the short answer is to populate a LogFont structure and pass it to the CreateFontIndirect API. CreateFontIndirect returns a font handle, which you can pass to SelectObject for use in any device context (see the Resources box for a caveat).

When you select the font into a picture box, by passing the font handle and the control's hDC property, you can then use native VB graphics methods—CurrentX, CurrentY, and Print—to draw with the new font. When you're finished, select the new font back out of the device context and destroy it (see the Resources box for more background).

The level of detail in these steps begs for encapsulation. I've written a class that wraps the creation and subsequent clean-up of logical fonts into a tidy black box (download Listing A from The Development Exchange; see the Download Free Code box at the end of the article for details). The CLogFont class accepts a StdFont object as the basic definition of the font you want. You can simply pass the picture box's Font property to CLogFont's LogFont property. The LogFont property's Set procedure assigns all the properties of the passed Font to a StdFont object local to the class, and calls the private CreateLogFont procedure to create a logical font. If you don't want to rotate the font, you can now read the CLogFont.Handle property and use it at will. To add a twist, pass the desired angle to CLogFont.Rotation, and a new log font is created using the passed value. Here's an example that prints text at a 45-degree angle after a mouse click:

```
Private Declare Function SelectObject Lib _
   "gdi32" (ByVal hDC As Long, ByVal hObject _
   As Long) As Long

Private fnt As CLogFont

Private Sub Form_Load()
   Set fnt = New CLogFont
   Set fnt.LogFont = Picture1.Font
   fnt.Rotation = 45
End Sub

Private Sub Picture1_MouseUp(Button As _
   Integer, Shift As Integer, X As Single, _
   Y As Single)
   Dim hFont As Long
   With Picture1
      hFont = SelectObject(.hDC, fnt.Handle)
      .CurrentX = X
      .CurrentY = Y
      Picture1.Print "Degrees: " & _
         fnt.Rotation
      Call SelectObject(.hDC, hFont)
   End With
End Sub
```

The Form_Load event instantiates the class and assigns its properties. The SelectObject API actually assigns the logical font to the picture box. SelectObject returns the handle of the font that was previously selected into the specified device context. When the

ABOUT THIS COLUMN

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, or ideas on the site, or access a comprehensive database of previously answered questions.

custom font is no longer needed, another call to SelectObject reassigns the picture box its previous font. This step is critical, because the logical font cannot be destroyed if it's still selected into any device context.

CLogFont handles all remaining dirty housekeeping chores. Whenever a new font is assigned, or a new angle requested, a call to DeleteObject destroys the existing logical font—if any—before a new one is created. When the Terminate event fires, it destroys the logical font similarly. Neglecting such clean-up is a classic example of what are called "resource leaks"—a misnomer, because no leak exists per se, just a failure by the programmer to put things back as they were found.

As you can see, the CreateFontIndirect API is rife with options. If these intrigue you, I'd encourage spending some time playing with them to observe how simple changes can have interesting impacts. Another powerful use for the CLogFont class is to support the Font property you expose in your own UserControls. If you write what's known as an owner-drawn control, this class provides the perfect link between the StdFont object requested by your client and the font handle required by the GDI calls you'll be making.

Q Dismissing a MsgBox After Time

I'd like to pop up a MsgBox, but have it close automatically if the user doesn't respond within a given time. I know I could create my own message box replacement form, but doesn't some API provide a shortcut?

A There's no magic bullet, but with just a little bit of trickery you can indeed coerce this result. Just before showing the MsgBox, set a Timer control to your desired time-out value and enable the timer. When the timer fires, use the FindWindow API to obtain the window handle of the message box, then call SendMessage to pass WM_CLOSE to it (see Listing 1). This results in the MsgBox returning the default cancel response. That is, it's exactly as if the user had popped the [X] button in the upper-right.

"But wait," you say, "message boxes block Timer events!" That *was* true up through VB4. And it remains true when you're running in the development environment. However, that's definitely not the case when you're using either VB5 or VB6 to compile your EXE. This change in behavior allows us to pull tricks that before were impossible. Experiment a bit to determine the value returned, and how to interpret it, if you're using multibutton message boxes.

Q Make Great Circle Calculations

I see you work with Geographic Information Systems (GIS). I am looking for the VB code, for use in an MS Access application, that computes the Great Circle distance between two given points expressed in latitude and longitude.

VB5, VB6 | Firing Timer Events During a MsgBox Call

```
Option Explicit

Private Declare Function FindWindow Lib "user32" _
  Alias "FindWindowA" (ByVal lpClassName As String, _
  ByVal lpWindowName As String) As Long
Private Declare Function SendMessage Lib "user32" _
  Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg _
  As Long, ByVal wParam As Long, lParam As Any) As Long

Private Const WM_CLOSE = &H10

Private Const MsgTitle As String = "Test Message"

Private Sub Command1_Click()
  With Timer1
    .Interval = 2000
    .Enabled = True
  End With
  MsgBox "I should disappear in two seconds.", , MsgTitle
End Sub

Private Sub Timer1_Timer()
  Dim hWnd As Long
  Timer1.Enabled = False
  hWnd = FindWindow(vbNullString, MsgTitle)
  Call SendMessage(hWnd, WM_CLOSE, 0, ByVal 0&)
End Sub
```

Listing 1 VB5 opened many new possibilities by allowing certain events to continue firing even while a message box is displayed. Here, a Timer event is employed to close a message box after a given time-out has expired without user interaction. The important thing to remember is that this only works from a compiled EXE, not from within the development environment.

A Well, this is one I've never needed before, but a quick Web search found just what you're looking for. It turns out many Web sites offer such calculations on their own forms, and a few have the algorithms posted as well. I found the one I chose to convert from CGI to VB at http://www.atinet.org/~steve/cs150/cs150.html (see Listing 2). Another great source of information on this topic is offered by the U.S. Census Bureau at http://www.census.gov/cgi-bin/geo/gisfaq?Q5.1, where you can find a discussion of the pros and cons behind the many choices made in coding up this algorithm.

The Distance function listed is actually part of a class I worked up to provide easy input of coordinates and desired scale units. Variables shown, but not local to the function, were set elsewhere in the class, but I believe the naming conventions should make their usage clear. As coded, Distance uses point locations expressed in decimal degrees. Converting from de-

RESOURCES

- Microsoft Knowledge Base article Q175535 points out an issue under VB5/SP2 with selecting a font into the Printer object's hDC. If you go this route, read the article and be sure to test.
- For more background on selecting a font into a picture box, see Microsoft Knowledge Base articles Q1154515 and Q119673.

When the timer fires, use the FindWindow API to obtain the window handle of the message box, then call SendMessage to pass WM_CLOSE to it.

VB4, VB5, VB6 | Calculate Great Circle Distance

```
Public Function Distance() As Double
  Dim L1 As Double           ' Start Point Latitude in radians
  Dim L2 As Double           ' End Point Latitude in radians
  Dim N1 As Double           ' Start Point Longitude in radians
  Dim N2 As Double           ' End Point Longitude in radians
  Dim C As Double            ' Cosine of the angle subtended by the
                             ' segment of the great circle path
                             ' between the two points
  Dim A As Double            ' Angle derived from C
  Dim R As Double            ' Radius of the Earth
  Const Pi As Double = 3.141592654

  ' Set radius to user's choice
  Select Case m_Units
    Case Kilometers
      R = 6378
    Case NauticalMiles
      R = 3444
    Case StatuteMiles
      R = 3963
  End Select

  ' Convert start/end points to radians
  L1 = m_Pt(Start).Latitude.Decimal * (Pi / 180)
  N1 = m_Pt(Start).Longitude.Decimal * (Pi / 180)
  L2 = m_Pt(Finish).Latitude.Decimal * (Pi / 180)
  N2 = m_Pt(Finish).Longitude.Decimal * (Pi / 180)

  ' Calculate C and A
  C = (Sin(L1) * Sin(L2)) + (Cos(L1) * Cos(L2) * Cos(N2 - N1))
  A = Atn(Sqr(1 - (C * C)) / C) + Pi * (C - Abs(C)) / (2 * C)

  ' Return A multiplied by the radius of the Earth
  Distance = A * R

End Function
```

Listing 2 This function is part of a CGreatCircle class, and provides the distance between any two points on a sphere. Alas, the Earth isn't a perfect sphere, so the calculation is at best a good approximation. All nonlocal variables are defined elsewhere in the class, which is available for registered users on The Development Exchange.

grees/minutes/seconds to this format works like this:

```
.Decimal = Sgn(.Degrees) * _
   (Abs(.Degrees) + _
   .Minutes / 60 + _
   .Seconds / 3600)
```

You can download the complete class from the free, Registered Level of The Development Exchange (see the Download Free Code box for details). Calculations such as this can never be entirely precise, so I urge a good deal of time spent surfing the Web and reading up on all the assumptions that go into such formulae, as well as considering the various arguments for and against each. VBPJ

About the Author

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, Wash., he's also an independent programming consultant specializing in ActiveX controls and contributes to various journals. Karl coauthored *Visual Basic 4 How-To* (Waite Group Press). Online, he's a Microsoft MVP, and a section leader several *VBPJ* online forums. Find more of Karl's VB samples at http://www.mvps.org/vb.

DOWNLOAD FREE CODE

Download the code for this issue of **VBPJ free** from http://www.vbpj.com.

To get the free code for this entire issue, type **VBPJ0199** into the Locator+ field at the top right of the VBPJ home page. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus the complete CLogFont and CGreatCircle classes.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0199AP** into the Locator+ field. The bonus code includes all the free code described above, plus example projects that demonstrate the CLogFont and CGreatCircle classes and another example that demonstrates timing out the different MsgBox styles.