Use Smart Text Tricks

Click & Retrieve Source CODE!

by Karl E. Peterson

Q Employ Custom Symbology in a Grid

I use a mapping package that employs characters from symbol-type fonts to display the position of point objects on maps. I want to write a VB program that can use a grid as the map legend, showing the object names together with their associated map symbols. I'll need to replicate the various options that the mapping package offers such as color, borders, and drop shadows for each point symbol. Even if I could do that, it seems that changing the font for the MSFlexGrid control is an all-or-nothing proposition. How can I apply these special effects to a symbol font character in one column, while using standard text in another?

A You're right about the MSFlexGrid control; it applies font assignments to the entire grid. The trick is to create a graphic containing the rendered point symbol, and assign this graphic to the CellPicture property of the grid control. The easiest way to do that is to add a hidden PictureBox control to your form, set its AutoRedraw property to True, draw the graphic on this PictureBox control, and assign the picture's Image property to the grid's CellPicture property.

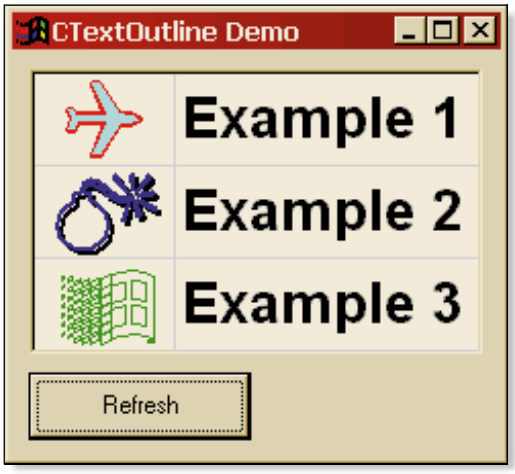

Figure 1 Use Special Symbol Effects in a Grid. This sample project uses the CTextOutline class to render symbols onto a hidden Picture control, which is then assigned to the CellPicture property of the grid. You can achieve a wide array of effects through creative use of color, outline, fill, and multiple overlays.

Rendering the graphic is a bit more of a challenge. Graphics Device Interface (GDI) paths are the answer. I've written a little class, CTextOutline, that encapsulates the process (see Listing 1 and Figure 1). CTextOutline exposes properties indicating whether the text should be filled or outlined, and optionally, whether the outline should fall behind the fill. In addition, CTextOutline exposes a device context handle (hDC) to use with the drawing functions. It's up to you to assign the desired GDI objects to the PictureBox, using its DrawWidth, FillColor, FillStyle, and ForeColor properties (see Listing 2).

Once you properly set up the picture box, call the DrawText method of CTextOutline, passing the string to render and the coordinates at which to start. DrawText starts by using SetBkMode to set the DC's background mode to transparent, storing the previous value so it can be restored later. Set the background to opaque instead if you prefer an inverse rendering.

The next operation is the heart of the solution. GDI supports path objects, which you can build using a variety of GDI calls. To create a path, call BeginPath, followed by the instructions needed to define the path, and finish with a call to EndPath. Use the GDI function TextOut to define your path as the outline of the passed string. To finish the rendering, call one or more of these GDI functions based on the property settings of the class: StrokePath, FillPath, or StrokeAndFillPath.

You can find an example project that uses the CTextOutline class on DevX (see the Download Free Code box for details).

Q Force a Fixed-Pitch Font

I'm writing an MDI application to act as an editor for a language I use. I would like the font to be a fixed type as in Notepad or the code editor in VB5.

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 www.inquiry.com/thevbpro. You can submit your questions, tips, or ideas on the site, or access a comprehensive database of previously answered questions.

RESOURCES

- "SAMPLE: How to Use Paths to Create Text Effects," Microsoft Knowledge Base Article ID: Q128091

How do I set the font to be fixed-pitch?

A You could go about this in one of several ways. If you'd like to offer the user a choice of what font to work with, you can apply the cdlCFFixedPitchOnly flag to the common dialog's Flag property:

```
Private Sub GetFont(txt As TextBox)
   With CommonDialog1
      .CancelError = True
      .Flags = cdlCFScreenFonts Or cdlCFFixedPitchOnly _
         Or cdlCFForceFontExist
```

VB4/32, VB5, VB6 | **Text Needn't Be Boring**

```
Option Explicit

Private Declare Function BeginPath Lib "gdi32" _
  (ByVal hDC As Long) As Long
Private Declare Function EndPath Lib "gdi32" _
  (ByVal hDC As Long) As Long
Private Declare Function TextOut Lib "gdi32" _
  Alias "TextOutA" (ByVal hDC As Long, ByVal X _
  As Long, ByVal Y As Long, ByVal lpString _
  As String, ByVal nCount As Long) As Long

Private Declare Function StrokeAndFillPath Lib _
  "gdi32" (ByVal hDC As Long) As Long
Private Declare Function StrokePath Lib "gdi32" _
  (ByVal hDC As Long) As Long
Private Declare Function FillPath Lib "gdi32" _
  (ByVal hDC As Long) As Long

Private Declare Function GetBkMode Lib "gdi32" _
  (ByVal hDC As Long) As Long
Private Declare Function SetBkMode Lib "gdi32" _
  (ByVal hDC As Long, ByVal nBkMode As Long) _
  As Long

' Background Modes
Private Const TRANSPARENT = 1
Private Const OPAQUE = 2

' Member variables
Private m_Filled As Boolean
Private m_hDC As Long
Private m_OutlineBehind As Boolean
Private m_Outlined As Boolean

' ******************************************************
'  Public Properties
' ******************************************************
Public Property Let Filled(ByVal NewVal As Boolean)
  m_Filled = NewVal
End Property

Public Property Get Filled() As Boolean
  Filled = m_Filled
End Property

Public Property Let hDC(ByVal NewVal As Long)
  m_hDC = NewVal
End Property

Public Property Get hDC() As Long
  hDC = m_hDC
End Property

Public Property Let OutlineBehind(ByVal NewVal As _
  Boolean)
  m_OutlineBehind = NewVal
End Property

Public Property Get OutlineBehind() As Boolean
  OutlineBehind = m_OutlineBehind
End Property

Public Property Let Outlined(ByVal NewVal As Boolean)
  m_Outlined = NewVal
End Property

Public Property Get Outlined() As Boolean
  Outlined = m_Outlined
End Property

' ******************************************************
'  Public Methods
' ******************************************************
Public Sub DrawText(ByVal Text As String, ByVal X As _
  Long, ByVal Y As Long)
  Static oldBkMode As Long
  Static nRet As Long

  If m_hDC Then
    oldBkMode = SetBkMode(m_hDC, TRANSPARENT)

    ' create the path within the DC
    Call BeginPath(m_hDC)
    Call TextOut(m_hDC, X, Y, Text, Len(Text))
    Call EndPath(m_hDC)

    If m_Outlined And m_Filled Then
      If m_OutlineBehind Then
        ' first draw the outline, then...
        Call StrokePath(m_hDC)
        ' recreate the path, then...
        Call BeginPath(m_hDC)
        Call TextOut(m_hDC, X, Y, Text, _
          Len(Text))
        Call EndPath(m_hDC)
        ' fill the path.
        Call FillPath(m_hDC)
      Else
        Call StrokeAndFillPath(m_hDC)
      End If
    ElseIf m_Filled Then
      Call FillPath(m_hDC)
    ElseIf m_Outlined Then
      Call StrokePath(m_hDC)
    End If

    Call SetBkMode(m_hDC, oldBkMode)
  End If
End Sub
```

Listing 1 You can render text using GDI path functions to create some stunning special effects. This code from the CTextOutline class draws a string based on various property settings of the class. The GDI Pen, Brush, and Font objects used in the rendering are selected using standard PictureBox properties.

VB4/32, VB5, VB6 | **Get Creative**

```
Private Sub Command1_Click()
  Dim txt As CTextOutline
  Dim char As String

  ' set up CTextOutline instance
  Set txt = New CTextOutline
  txt.hDC = Picture1.hDC

  ' draw an blue-outlined character with shadow
  With Picture1
    .Cls
    char = Chr(Rnd() * 235 + 20)
    .CurrentX = (.ScaleWidth - .TextWidth(char)) \ 2
    .CurrentY = (.ScaleHeight - .TextHeight(char)) \ 2
  .DrawWidth = 2
    .ForeColor = vb3DDKShadow
    txt.Filled = False
    txt.Outlined = True
    txt.OutlineBehind = False
    txt.DrawText char, .CurrentX + 2, .CurrentY + 2
    .ForeColor = vbBlue
    txt.DrawText char, .CurrentX, .CurrentY
  End With
  With MSFlexGrid1
    .Row = 1
    .Col = 0
    .CellPictureAlignment = flexAlignCenterCenter
    Set .CellPicture = Picture1.Image
  End With
End Sub
```

Listing 2 Although the API calls are encapsulated behind a class, lots of settings are available if you want to get really creative with CTextOutline. This example overlays two renderings of the same symbol, first creating the shadow effect followed by the actual symbol, and finally assigning it to a grid cell for display (see Figure 1).

```
      On Error Resume Next
      .ShowFont
      If Err.Number = 0 Then
         txt.Font.Name = .FontName
         txt.Font.Size = .FontSize
      End If
   End With
End Sub
```

This approach assumes the user indeed has some fixed-pitch fonts installed—a fairly solid assumption. You can also use stock font objects supplied directly by Windows:

```
Private Sub GetStockFont(txt As TextBox)
   Dim hFont As Long
   hFont = GetStockObject(ANSI_FIXED_FONT)
   Call SendMessage(txt.hWnd, WM_SETFONT, hFont, False)
   Call DeleteObject(hFont)
End Sub
```

I recommend this second route at application startup. Then you should provide an option for the user to call the common Fonts dialog to choose another font, and store a reference to this preferred font for future runs.

Q Clone Font to Break Connection

I've observed that if I set an object's Font property to that of another object's Font property, any later change to either object's Font property affects both. In other words, the two objects share references to a common Font object. This is well and good in most cases, but other times I'd like to break this connection—to assign a clone of another object's Font property. Is there any way to do this short of assigning each subproperty (Name, Bold, Size, and so on) individually?

A In fact, yes, there's an extremely simple method—one that I include in nearly every project I work on. Look at this:

```
Function FontClone(fnt As IFont) As StdFont
   fnt.Clone FontClone
End Function
```

The IFont interface is implemented by VB's StdFont class, so you can pass any font reference directly to this function. The cloned font is assigned to the new StdFont class instance, which you passed to the Clone method of the IFont object. Wrapping this logic up into a function allows you to clone any Font reference as simple as:

```
Set Text1.Font = FontClone(Me.Font)
```

The result is that the two objects' Font properties refer to different StdFont objects, so changing one won't change the other anymore. This technique can also be especially handy in classes and user controls that expose a Font property. 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. Online, he's a Microsoft MVP and a section leader on several **VBPJ** forums. Find more of Karl's VB samples at www.mvps.org/vb.

DOWNLOAD FREE CODE

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

To get the free code for this entire issue, click on Locator+, the right-most option on the menu bar at the top of the home page, and type **VBPJ0799** into the box. (You first need to register, for free, on DevX.) The free code for this article includes all code listings, plus a sample containing the CTextOutline class (shown in Figure 1), and a simple text editor that forces use of a fixed-pitch font.

✪ To get the bonus code for this article, available to DevX Premier Club members, type **VBPJ0799AP** into the Locator+ field. The bonus code includes all the free code described above, plus a sample app that includes an extended CTextOutlineEx class that will optionally create GDI objects for you and rotate the rendering to any angle.