PROGRAMMING TECHNIQUES

Click & Retrieve Source CODE!

# Making Progress

## *Build your own Progress Bar class to use in both 16- and 32-bit versions of VB4.*

by Karl E. Peterson

Windows 95 and Windows NT offer many new standard controls for Visual Basic programmers. Unfortunately, you cannot use these controls in the 16-bit version of VB4, even when running in 32-bit operating systems. This month I will present a VB4 class module that re-creates the new ProgressBar control. You can use it in both 16- and 32-bit versions of VB4, or modify the same general technique to operate in VB3, without regard to which operating system your application is running under.

In the *VBPJ* January 1996 Programming Techniques column, I showed how to add capabilities to an existing control by encapsulating "type-amatic" searches for list boxes within a class module. In this month's technique, a class module empowers one standard control to emulate another. Hopefully, the finished class also will prove useful for those of you writing 16-bit code, or conditionally compiling to both 16- and 32-bit executables. Those of you who've already committed completely to 32-bit may wonder if there's any value in re-creating the ProgressBar. Consider that it's one less OCX you'll need to distribute if your project doesn't use any of the other Windows 95 common controls.

The general idea behind the CProgressBar class is that a picture box on a form will be used as a canvas upon which the class can draw a progress indicator. You will need a number of private member variables within the class to track its internal data. After you insert a new class module in your project, enter these declarations in its General section:

```
'
' Set aside storage for private member
' variables.
'
Private m_Bar As PictureBox
Private m_Val As Long
Private m_Min As Long
Private m_Max As Long
Private m_fColor As Long
Private m_bColor As Long
Private m_CellWidth As Integer
Private m_CellHeight As Integer
Private m_xMargin As Integer
Private m_yMargin As Integer
Private m_Cells As Integer
```

Use the class' Initialize event to set default values into several of the member variables. Set the default minimum and maximum property values to match those of the real ProgressBar control, zero and 100. Likewise, set the initial value to zero. Set m_Bar, the canvas upon which drawing will take place, to Nothing because you do not know what picture box you will use at this point. Set the foreground and background colors to match the operating system's default colors. Use the x and y margin values to calculate the number and size of cells to draw.

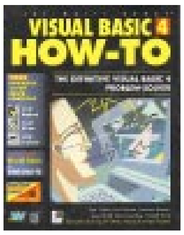
```
Private Sub Class_Initialize()
   '
   ' Set default values for class
   ' properties.
   '
   Set m_Bar = Nothing
   m_Val = 0
   m_Min = 0
   m_Max = 100
   m_fColor = vbHighlight
   m_bColor = vb3DFace
   m_yMargin = 2
   m_xMargin = 3
End Sub
```

The CProgressBar class exposes a Canvas property, which you will use to set or retrieve the picture box object used for the emulated progress bar (see Listing 1). Download PT0396.ZIP for demo projects using the CProgressBar class, as well as other code from this column, from the Magazine Library of the *VBPJ* Forum on CompuServe. In the Property Set routine, the new value for this property is received As Object. This allows the class to detect what type of object the routine passed, should an object besides a picture box be assigned to the Canvas property.

If a picture box was received, a reference to it is stored in the m_Bar member variable, and new ForeColor and BackColor values are assigned to it. You also set it to use Pixels as its ScaleMode, so that you can perform later drawing operations more easily. If the passed object is not a picture box, the routine generates a runtime error with appropriate explanation, and the client application is left to handle it.

For consistency, use standard property names for your classes, especially when they're emulating something else. The

*Karl E. Peterson is a GIS analyst with a regional transportation planning agency in Vancouver, Wash., and a member of the* VBPJ *Technical Review Board. Karl coauthored* Visual Basic 4 How To, *from Waite Group Press. Online, he's the 32-Bit Bucket section leader in the* VBPJ *Forum and a Microsoft MVP in the MSBASIC Forum. Contact Karl in either CompuServe location at 72302,3707.*



Min, Max, and Value properties each have their respective Let and Get routines (these listings are posted in the *VBPJ* Forum). Property procedures have an advantage over Public variables in that you can validate the incoming values. For these properties, the rules are simple and follow those used by the real ProgressBar control. The Min must be less than the Max, and the Max must be greater than the Min. If you do not meet either of these conditions, the routine generates a runtime error, with an explanation detailing the source of the error and its cause. The Value property must fall somewhere between the Min and the Max, but is simply set to either extreme should an out-of-range value be received. When a client application alters any of these three properties, that Let procedure calls the public Refresh method.

The Refresh method repaints the emulated progress bar, and may be called directly by the client application. If the optional parameter ClearFirst is set to True, the Canvas (m_Bar) will first be erased using its Cls method. Then, the private RedrawMe procedure is called to do the actual drawing of the progress bar cells:

```
Public Sub Refresh(Optional ClearFirst)
   '
   ' Update display, clearing
   ' it if requested.
   '
   If Not m_Bar Is Nothing Then
      If Not IsMissing(ClearFirst) Then
         If ClearFirst Then
            m_Bar.Cls
         End If
      End If
      RedrawMe
   End If
End Sub
```

VB4

```
Public Property Set Canvas(NewObj As Object)
  '
  ' Set new PictureBox as Canvas property.
  '
  If TypeOf NewObj Is PictureBox Then
     Set m_Bar = NewObj
     m_Bar.ForeColor = m_fColor
     m_Bar.BackColor = m_bColor
     m_Bar.ScaleMode = vbPixels
     ResizeEx
  Else
     Err.Raise Number:=vbObjectError + 1, _
              Source:="CProgressBar.Canvas", _
              Description:="Canvas property must be _
              of type PictureBox."
  End If
End Property

Public Property Get Canvas() As Object
  '
  ' Return PictureBox as Canvas property.
  '
  Set Canvas = m_Bar
End Property
```

LISTING 1 ***Preparing the Class' Canvas.*** *The Canvas property sets and returns the picture box you will use for your custom progress bar.*

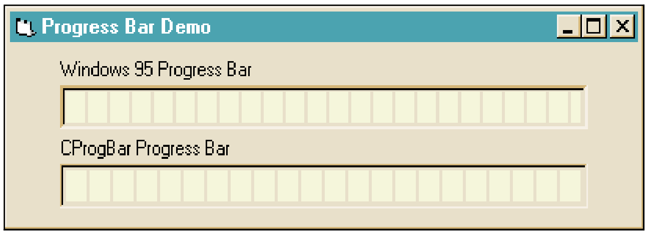


FIGURE 1 ***CProgressBar "Control" Outsmarts the Win95 Custom Control.*** *This illustration shows how the CProgressBar class is actually smarter than the Windows 95 custom control. By using the ResizeEx method, the programmer specifies that an exact number of whole indicator cells are drawn, rather than accepting the default behavior of a partial last cell as Windows 95 does.*

Before any drawing can take place, you need to make a number of calculations. You may have noticed that a ResizeEx method was called from the Set Canvas procedure. The ResizeEx method, being public, is also available to the client application. ResizeEx calls the CalcCellSize private method, which provides the dimensions for each cell and the number of cells required to fill the progress bar (this method is posted in the *VBPJ* Forum). It uses margins defined in the class Initialize event, and a 2:3 width-to-height ratio to determine individual cell sizes. So far, I've directly emulated the real ProgressBar control.

ResizeEx takes things one step further, though. The real ProgressBar control behaves in a non-integral manner, often using a partial cell to indicate final completion (see Figure 1). ResizeEx determines whether this last partial cell is wider or narrower than half a whole cell. If wider, ResizeEx resizes the Canvas object again, enlarging it to include the entire last cell. If narrower, ResizeEx shrinks the Canvas object to exclude the last cell, and it decreases the number of cells to paint by one.

VB4

```
Private Sub RedrawMe()
  Dim i As Long
  Dim x As Long
  Dim y As Long
  Dim n As Long
  '
  ' Calc number of live cells to draw.
  '
  n = (m_Val / (m_Max - m_Min)) * m_Cells
  '
  ' Draw live cells.
  '
  m_Bar.ForeColor = m_fColor
  y = m_yMargin + m_CellHeight - 1
  x = m_yMargin
  For i = 1 To n
     m_Bar.Line (x, m_yMargin)-_
        (x + m_CellWidth, y), , BF
     x = x + m_xMargin + m_CellWidth
  Next i
  '
  ' Draw dead cells.
  '
  If n < m_Cells Then
     m_Bar.ForeColor = m_bColor
     For i = n + 1 To m_Cells
        m_Bar.Line (x, m_yMargin)-_
           (x + m_CellWidth, y), , BF
        x = x + m_xMargin + m_CellWidth
     Next i
  End If
End Sub
```

LISTING 2 ***Drawing the Emulated Progress Bar.*** *This private method is called from within the CProgressBar class whenever the progress bar needs to be refreshed. It first draws "live" cells to indicate completion progress, then draws "dead" cells using the background color to erase the remainder of the progress bar.*

Because the Canvas object will likely be resized, you must prevent recursion in case its Picture_Resize event itself calls the ResizeEx method. You can do this by setting a Static variable to True immediately before resizing the Canvas object. Upon entering the ResizeEx method, if this Static variable is found to be True, it is then assumed that a recursive call has been made, and the method is exited without further action. After the Canvas object has been resized, the Refresh method is called to repaint it using its current Value.

A private RedrawMe method is called from the public Refresh method, and this is where the actual drawing takes place (see Listing 2). Knowing the cell dimensions, you need just one calculation to determine the number of "live" cells—those in the highlight color—to be drawn based on the Value property. It's then a fairly simple task of drawing each cell using VB's Line method with the filled box option.

First, the program draws each "live" cell in the foreground color, then it draws the remaining "dead" cells using the background color. The extra "dead" cells need to be drawn because progress may not be positive in all cases. A progress bar may not in fact move steadily from zero to 100, but may be monitoring something where occasional setbacks occur. In these cases, the program must now redraw the previously "live" cells as "dead."

Using the CProgressBar class in your projects is extremely simple; it requires only a few more steps than are required when using the ProgressBar control itself. When designing the form, place a picture box where you want the progress bar to be. In the General section of the form, include this declaration to create a new instance of CProgressBar named pBar:

```
Private pBar As New CProgressBar
```

Then, in the Form_Load event, pass a reference to the picture box as the Canvas property of the CProgressBar instance:

```
Set pBar.Canvas = Picture1
```

From this point on, you can call CProgressBar's methods and set its properties at will. For example, to run it through its paces, use a loop such as this:

```
For i = 0 To 100 Step 4
   pBar.Value = I
   ‘ Win32 API function to delay
   ‘ 100 milliseconds
   Sleep 100
Next I
pBar.Value = 0
```

If your form is resized, and you want to resize the progress bar to fit the new space, first resize the picture box then call the ResizeEx method for an integral number of cells (if exact fit is more important than the distraction of a partial last cell, CProgressBar also contains a Resize method that simply recalculates cell dimensions, but doesn't adjust the Canvas object's size). You can accomplish this most easily by placing the call to Resize(Ex) in the Picture_Resize event. Likewise, a call to the Refresh method would be prudent within the Picture_Paint event. ☒