Move the Mouse Cursor in Code

Have you ever wanted to move the mouse cursor under program control? This tip shows you how to do so using the API function SetCursorPos. The declaration for this function is:

```
Declare Function SetCursorPos Lib "user32" (ByVal x As Long, _
   ByVal y As Long) As Long
```

The two arguments give the cursor position. This must be given in screen pixels, so there's a twip-to-pixel conversion required when using the function in a Visual Basic program. The return value, which is often ignored, is 0 on failure and nonzero on success.

Why would you want to move the mouse cursor under program control? The main reason that I use this technique is to position the cursor at the location where the user is most likely to want it. For example, if your program displays a form for confirmation of program settings, and the user is most likely to simply accept the default settings by clicking the OK button, you could position the mouse cursor over the OK button when the form is displayed, saving the user the effort.

To use the SetCursorPos function you obviously must know the pixel coordinates of the location that you want to move the cursor to. Suppose, for example, you want the cursor positioned in the center of a Command Button. At first glance the following calculations seem to be what's needed (these are for the X position; the Y position follows the same logic):

(The form's Left property) plus (the control's Left property) plus (one-half the control's Width property)

This is on the right track, but it ignores the width of the form borders and, for the Y coordinate, the title bar. Remember, the Form.Left property gives the position of the outer corner of the form, while a control's Left property gives its position relative to the form's client area (the interior). To determine the width of the form's border you can subtract the form's ScaleWidth, the width of the interior area, from the Width (the width of the entire for including borders). By adding this value to the calculation above you will be right on target. Note that this technique requires that the form's ScaleMode be at the default Twips setting, otherwise it will not work.

Wait, we are not quite done - this calculation gives the desired cursor position in Twips while the SetCursorPos function needs pixels. This is easily remedied by dividing with the Screen.TwipsPerPixelX and Screen.TwipsPerPixelY properties to convert twips to pixels.

The sample code below shows how this is done. This is a form's Load event procedure. When the form is loaded, the mouse cursor is automatically positioned on a Command Button control named Command1.

```
Private Sub Form_Load(sender As Object, e As EventArgs) Handles MyBase.Load
     Dim x As Long, y As Long

     x = ((Me.Left + (Me.Width - Me.ScaleWidth) + _
       Command1.Left) + Command1.Width / 2) _
       / Screen.TwipsPerPixelX

     y = ((Me.Top + (Me.Height - Me.ScaleHeight) + _
       Command1.Top) + Command1.Height / 2) / _
       Screen.TwipsPerPixelY

     SetCursorPos(x, y)
```

End Sub

If you choose to add this feature to your program, it's advisable to make it an option that the user can turn on or off. Some users love it, others hate it, but this way you can please everyone!