CS109
PictureBox and Timer Controls

Let's take a little break from arrays and discuss two new controls. The picturebox control is used to display an image on the form. The Timer control is used to generate events in time intervals. Both controls are described briefly in chapter 9.

The PictureBox control is used to display graphics. It can display shapes we draw ourselves and also common image formats such as JPG, GIF, BMP, etc.

Drawing Line Graphics

To experiment with drawing graphics within a picturebox, add a PictureBox control to the form. In the VB.NET Code section, go to the events for the PictureBox and select the Paint event:

The Paint event is automatically invoked whenever the PictureBox needs to be redrawn. For example, if the form is minimized, dragged, or occluded, then when the form is activated then the Paint event will be invoked. By placing the drawing code in the Paint event it will always be updated correctly. If we placed the code somewhere else and the window was obscured, it may not be redrawn correctly when the obscuring item is moved out of the way.

We can now put code here that will draw whatever we like on top of the form. Just like in standard graphics format, the origin is the upper left corner. The x coordinates increase to the right, and the y coordinates increase down toward the bottom.

Here is some sample code we can add to the Paint event to draw various shapes on the screen:

```
Private Sub PictureBox1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles PictureBox1.Paint
   Dim g As Graphics
   ' Get the graphics object for the event (i.e. the PictureBox)
   g = e.Graphics

   ' Draw a red rect of width 1 at Width=50, Height=80 at coord 1,20
   g.DrawRectangle(Pens.Red, 10, 20, 50, 80)

   ' Make a new pen of width 4
   Dim thickPurplePen As Pen = New Pen(Color.Purple, 4)
   ' Ellipse in purple, width 4, within bounding rectangle at 50,10
   g.DrawEllipse(thickPurplePen, 50, 10, 40, 30)

   ' Draw a line from 10,10 to 50,50 of width 1
   g.DrawLine(Pens.MediumSeaGreen, 10, 10, 50, 50)

   ' To fill in a shape we must use a brush
   Dim bru As New SolidBrush(Color.GreenYellow)
   ' Fill in the rectangle
   g.FillRectangle(Brushes.GreenYellow, 100, 100, 50, 20)

   ' Draw part of a pie
   g.FillPie(Brushes.IndianRed, 130, 20, 100, 100, 30, 60)

   ' Draw the text "Abstract Art" in font Arial, size 12, in Indigo
   g.DrawString("Abstract Art", New System.Drawing.Font("Arial", 12), _
      Brushes.Indigo, 50, 140)
End Sub
```

First, we capture the Graphics object from the event arguments. The Graphics object is attached to the Form, so anything we draw on the graphics object will be displayed on the entire Form. We could attach code to the Paint event of specific controls to only paint within those controls, if we wish.

Next, we create a red pen and draw a rectangle using that pen. The rectangle takes the coordinates of the upper left corner then the width and height.

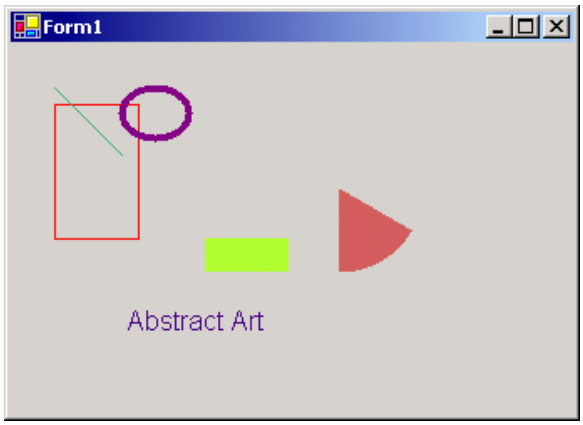
An ellipse is drawn in a similar fashion, by specifying the bounding rectangle that holds the ellipse. This time we create a new pen object. The new pen object can specify the width of the item to draw.

Next we draw a single line using the MediumSeaGreen pen.

Next we draw a solid rectangle using a Brush object. The Brush in this case is a solid color, but it is possible to create brushes that are hatched, texture, etc. Finally we draw part of a Pie slice using a red brush.

Finally we draw text toward the bottom of the screen using DrawString. You can pick whichever font you like that is on the system.

The picture created is shown below:



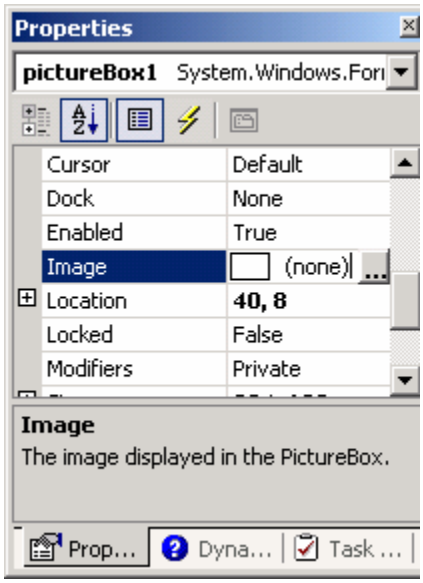
There are many other drawing tools available; see the text and online help for more details.

Note that we should always draw the items in the Paint event, or the items won't be refreshed properly if the screen needs to be re-drawn. For example, if the above code was placed in a Button Click event, the items would be drawn when the button is clicked but not when the form needs to be refreshed.

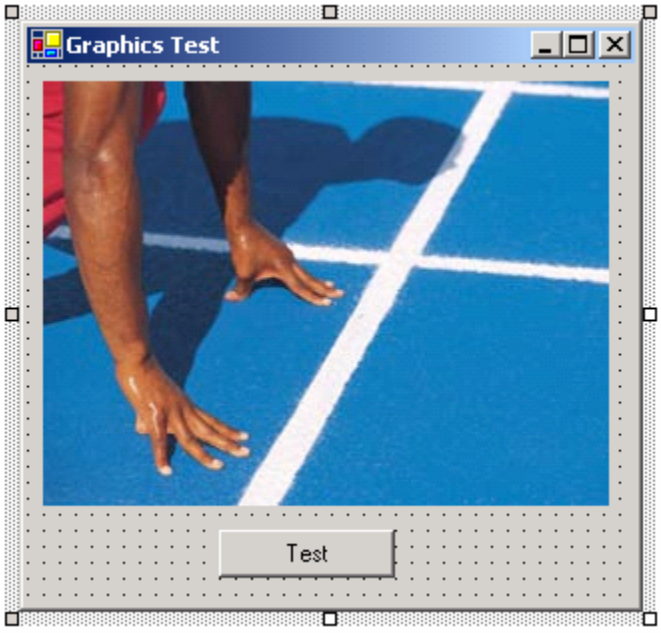
Working with Images

A pictureBox can also be used to display stored graphical images, such as JPG, GIF, BMP, PNG, and many other file formats. For example, such images might be created in a Paint Program and stored on the disk.

To place a static image in a picture box, drag the picture box onto the form and then select the "Image" property:



Click the "…" which will bring up a file dialog to search for an image. Find some image on your system. It should then be displayed inside your picturebox control. You may need to resize the control so that the entire image fits. In the picture below, I have selected the "sample.jpg" image that comes with Windows 2000 in the "My Documents/My Pictures" folder.



At this point you can run the application and it will display an image on the form. This is all you would need to do if you want to include static images in your application. The image data will be stored as part of the assembly of the executable, so you don't need to include the actual image file with your application.

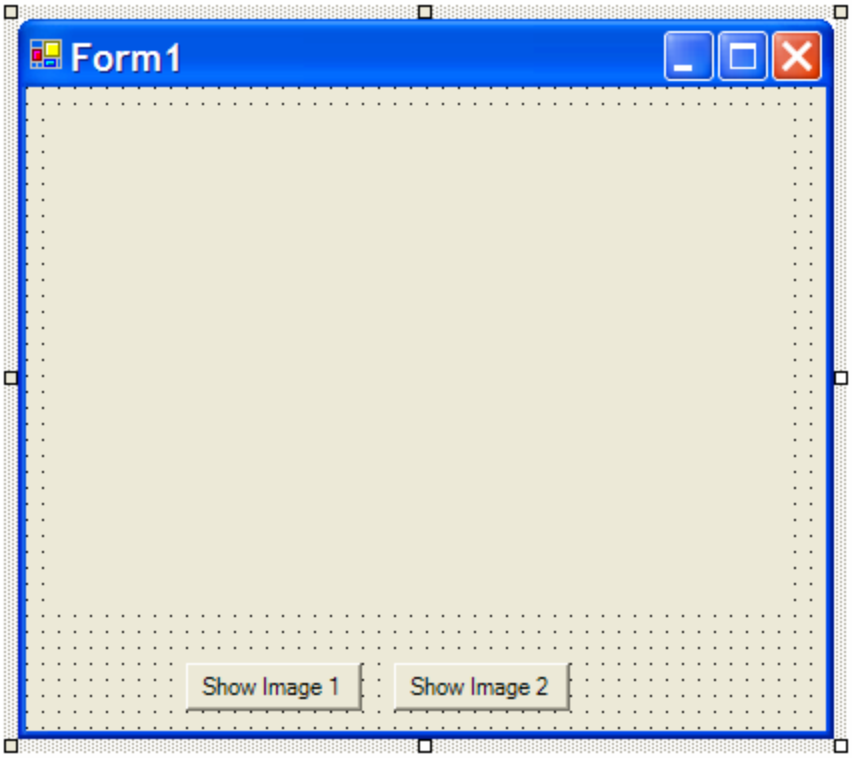
Next let's look at how you can programmatically set what is displayed in the picture box.

We can load an image from disk by setting the Image property of the picturebox to an image object. Here is an example:

```
PictureBox1.Image = Image.FromFile("c:\myimage.jpg")
```

When this code is executed, it will load the file "c:\myimage.jpg" and display it in the image.

If a lot of images are going to be displayed, a common technique is to load all of the images into memory when the program starts, and then display the loaded images later when required by the program. To do this we can create an Image object that can store a loaded image from disk. The following example shows how to load two images, and then display them when the appropriate button is clicked:



The form above has a picturebox in the middle and two buttons.

The code looks like the following and assumes that there are two files on the disk named “picture1.bmp” and “picture2.bmp”

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim image1, image2 As Image

    ' Load images from disk into variables when the form is started
    Private Sub Form1_Load(…) Handles MyBase.Load
        image1 = Image.FromFile("c:\picture1.bmp")
        image2 = Image.FromFile("c:\picture2.bmp")
    End Sub

    ' Display image1
    Private Sub btnShow1_Click(…) Handles btnShow1.Click
        PictureBox1.Image = image1
    End Sub

    ' Display image2
    Private Sub btnShow2_Click(…) Handles btnShow2.Click
        PictureBox1.Image = image2
    End Sub
End Class
```

Since class variables are accessible in any subroutine in the form, we first load the images into two class variables when the program starts. Then to display them we set the Image property of the picturebox to the appropriate class variable.

If we had a lot of images to load, we could use an array of Images instead. Here is an example for our array of two images:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim imageArray(2) As Image

    Private Sub Form1_Load(…) Handles MyBase.Load
        imageArray(1) = Image.FromFile("c:\picture1.bmp")
        imageArray(2) = Image.FromFile("c:\picture2.bmp")
    End Sub

    ' Display image1
    Private Sub btnShow1_Click(…) Handles btnShow1.Click
        PictureBox1.Image = imageArray(1)
    End Sub

    ' Display image2
    Private Sub btnShow2_Click(…) Handles btnShow2.Click
        PictureBox1.Image = imageArray(2)
    End Sub
End Class
```

If we had 20 images, all named “picture1.bmp” up to “picture20.bmp” we could display them as shown below.

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim imageArray(20) As Image
    Dim whichImage As Integer

    Private Sub Form1_Load(…) Handles MyBase.Load
        Dim i As Integer
        whichImage = 1      ' The current image to show

        For i = 1 To 20
            imageArray(i) = Image.FromFile("c:\picture" & CStr(i) _
                  & ".bmp")
        Next
        PictureBox1.Image = imageArray(1)   ' Show image 1 now
    End Sub

    ' Display next image
    Private Sub btnShow1_Click(…) Handles btnShow1.Click
        whichImage = whichImage + 1
        If (whichImage > 20) Then
            whichImage = 1  ' Cycle back to first image if on last one
        End If
        PictureBox1.Image = imageArray(whichImage) ' Display It
    End Sub
End Class
```

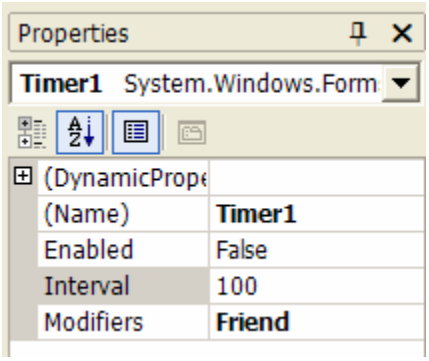
In the Form1_Load event, we are using a loop to load twenty image files. The names of the files are constructed by the loop; it assumes each is named picture<NUM>.bmp. To keep track of what image is being displayed, there is a class variable called whichImage. This variable is set to the index in the array of the current image being shown.

In the button click event, we increment the whichImage variable to move to the next image. If we have shown the last one (i.e. we now exceed the number of images) then cycle back to show the first image.

Timer Control

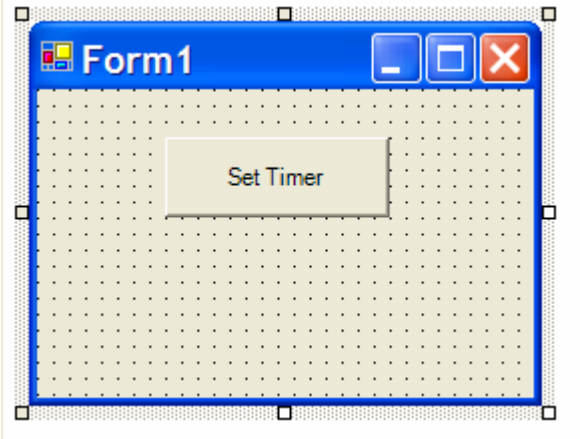
The timer control is used to execute code after some time interval has passed. Unlike all of the other controls we have used so far, it is invisible at runtime. VB.NET places invisible controls in a pane at the bottom of the screen instead of within your form.

The properties of the timer control are shown in the figure below:



The timer is inactive if Enabled is set to False. Once Enabled is set to true, the timer behaves like an alarm clock. It starts counting down, using the value in the Interval property, until it reaches zero. The value stored in the Interval property is in **milliseconds**, not seconds, so if you wanted a countdown of 5 seconds then you would store a value of 5000 in the Interval property.

The event that occurs when the timer reaches zero is the Timer1.Tick event. You can add code to this event by double-clicking on the Timer control. The following example initializes the timer and then displays a message after five seconds are up:

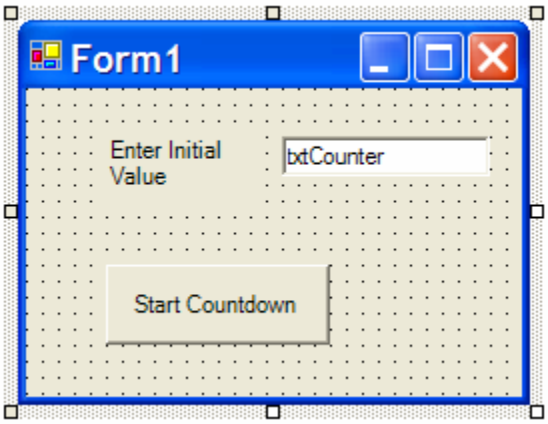



```
Private Sub Timer1_Tick(…) Handles Timer1.Tick
    ' Display a message
    MsgBox("Beep!")
End Sub

Private Sub Button1_Click(…) Handles Button1.Click
    Timer1.Interval = 5000  ' 5000 milliseconds
    Timer1.Enabled = True   ' Start the countdown!
End Sub
```

After a timer "goes off" it is still enabled and resets itself back to the interval value. In the above example, every five seconds a message box will be displayed. If we ever want to disable the timer, we can just set the Enabled property to false.

Here is another example that displays a countdown, in seconds, until we reach zero:



```
Private Sub Button1_Click(…) Handles Button1.Click
    Timer1.Interval = 1000          ' 1 second intervals
    Timer1.Enabled = True           ' Start countdown
End Sub

Private Sub Timer1_Tick(…) Handles Timer1.Tick
    Dim count As Integer

    count = CInt(txtCounter.Text)   ' Get current count value
    count -= 1          ' Subtract one
    txtCounter.Text = CStr(count)   ' Display new count
    If (count = 0) Then         ' Time expired
        Timer1.Enabled = False  ' Turn timer off
        MsgBox("Beep beep!")
    End If
End Sub
```

How could we make the countdown go twice as fast?

What might happen if we switch the order of statements in the If statement to:

```
    If (count = 0) Then         ' Time expired
        MsgBox("Beep beep!")
        Timer1.Enabled = False  ' Turn timer off
    End If
```

Can you figure out what is going on?