CS109

PictureBox and Timer Controls

Let's take a little diversion and discuss how to draw some simple graphics. Graphics are not covered in the book, so you'll have to use these notes (or the built-in help) as a reference.

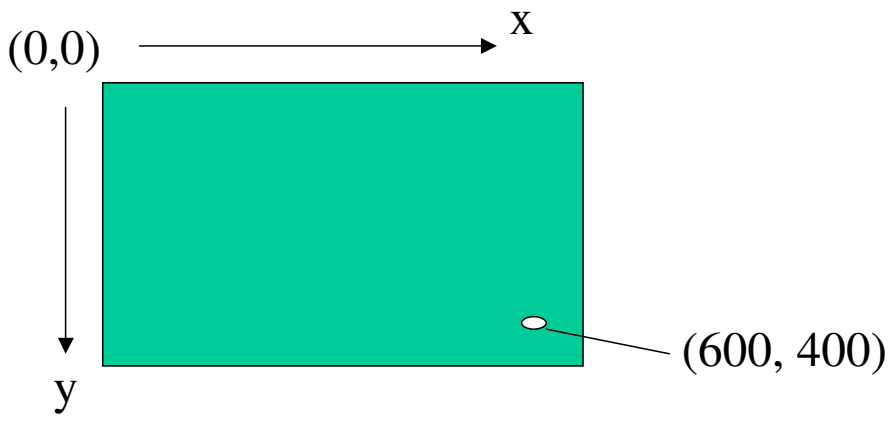
Generally, you will use the PictureBox control to display graphics. It can display shapes we draw ourselves and also common image formats such as JPG, GIF, BMP, PNG, etc. We already showed how to display a static image (just add a picturebox to the form, click on the image property, Import a local resource, and pick the file that corresponds to the image you want displayed).

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 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 PictureBox. The graphics screen is set up as a grid of pixels where the upper left coordinate is 0,0. This is relative to where the PictureBox is on the form. The x coordinate then grows out to the right, and the y coordinate grows down toward the bottom. For example, in the picture below the white pixel is at coordinate (600,400).



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 10,20
   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 required to draw graphics on the screen. Remember that everything will be drawn relative to the upper-left corner of the PictureBox.

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.

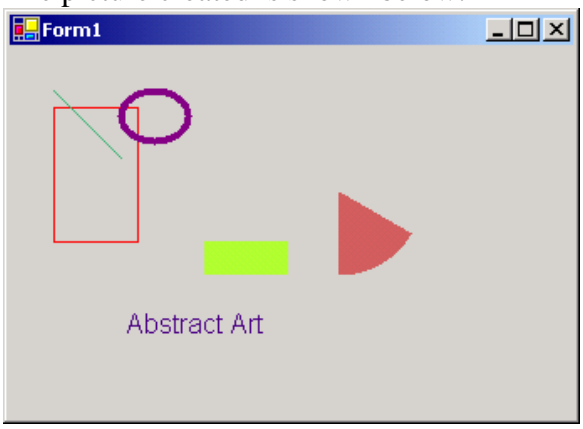
An ellipse is drawn in a similar fashion, by specifying the bounding rectangle that holds the ellipse. This time we create a new pen object. The new pen object can specify the width of the item to draw.

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 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.

Colors

There are lots of different pre-defined colors available from the Color object, e.g.:

Color.Black	Color.DarkGray	Color.Gray
Color.Blue	Color.Green	Color.LightGray
Color.Cyan	Color.Magenta	Color.Orange
Color.Pink	Color.Red	Color.White

To create our own color, we can specify the color we want in RGB (red, green, blue). To do this, use:

```
Color.FromArgb(Red, Green, Blue)
```

where Red, Green, and Blue are values in the range 0-255. 0 is the darkest intensity of each color and 255 is the brightest intensity of that color. The colors are mixed somewhat like the colors that make up light, e.g. red + green = yellow.

For example:

new Color(0,255,200);

Creates a green-blue color, with stronger green than blue. There is no red since its value is 0.

In-Class Exercise: Random rectangles

Write a program so that each time a button is pressed, the picturebox on the form displays a rectangle with a random width between (10-400) and a random height between (10-400) in a random color (SolidBrush) in a random position.

Use picturebox1.height and picturebox1.width to get the width and height of the picturebox in pixels. Rnd.next(X,Y) gives a random integer between X and Y.

To force the picturebox to update itself, inside the button click event add the code:

```
PictureBox1.Invalidate()
```

This invokes the Paint subroutine for PictureBox1 and it will re-draw itself by executing the code in the Paint event.

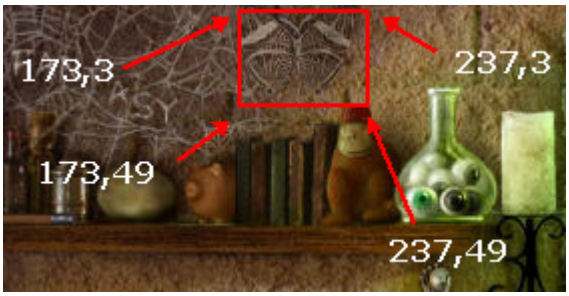
Image MouseClick Event

If you want to know where a user is clicking on an image, you can access the MouseClick event. Let's make an example using an image from Scholastic's I Spy game. Below is a picturebox named pboxISpy that has the following image loaded into it:

Let's make a simple game where the user has to click on the two butterflies hidden in the picture. We can do this by finding out the coordinates for rectangles that surround each

butterfly. If the user clicks in those coordinates then we know that they've clicked on a butterfly.

To find the coordinates we could use a paint program, like Paint. As you move the mouse on the image, the coordinates are displayed in the status bar:



For the top butterfly, if the user clicks anywhere where the X coordinate is between 173 and 237 and the Y coordinate is between 3 and 49, then the user will have clicked on the top butterfly.

Similarly, if the user clicks anywhere where the X coordinate is between 160 and 229 and the Y coordinate is between 393 and 417, then the user will have clicked on the bottom butterfly.

Another way to find these coordinates, other than by using a paint program, is to output the coordinates from Visual Basic. Put the following in the MouseClick event for the picturebox:

```
Private Sub pboxISpy_MouseClick(. . .) Handles pboxISpy.MouseClick
    Console.WriteLine(e.X & " " & e.Y)
End Sub
```

Now, run the program. If you click in the picturebox, the Output window will show the coordinates that you are clicking on. You could then click on various points of interest on the image to find their coordinates.

Here is a program that goes into the MouseClick event that then pops up a messagebox if the user found one of the butterflies:

```
    Private Sub pboxISpy_MouseClick(. . .) Handles pboxISpy.MouseClick
        If (e.X > 173) And (e.X < 237) And _
           (e.Y > 3) And (e.Y < 49) Then
            MessageBox.Show("You found the upper butterfly!")
        ElseIf (e.X > 160) And (e.X < 229) And _
           (e.Y > 393) And (e.Y < 417) Then
            MessageBox.Show("You found the lower butterfly!")
        End If
    End Sub
```

Finally, let's modify our game so it can keep track of how many items we found. If we find both butterflies then we will exit the game.

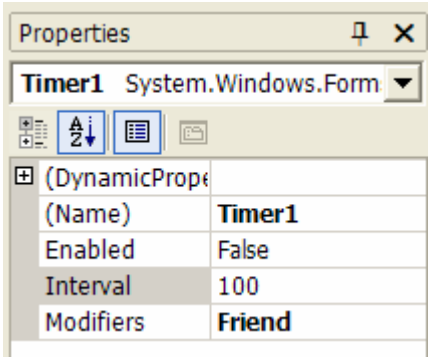
To do this, we'll create new class-level variables. Boolean variables will keep track of which butterflies have been found. This will prevent a message from popping up if a user clicks on the same butterfly twice. A counter variable will keep track of how many total items we've found. If we ever find two items then we exit. The program should be straightforward to extend to all items in the image, if desired.

```
Public Class Form1
    Dim intNumFound As Integer = 0
    Dim blnFoundUpperButterfly As Boolean = False
    Dim blnFoundLowerButterfly As Boolean = False


    Private Sub pboxISpy_MouseClick(. . .) Handles pboxISpy.MouseClick
        If (e.X > 173) And (e.X < 237) And _
           (e.Y > 3) And (e.Y < 49) Then
            If (blnFoundUpperButterfly = False) Then
                MessageBox.Show("You found the upper butterfly!")
                intNumFound += 1      ' Increment number of items found
                blnFoundUpperButterfly = True  ' Set butterfly found
            End If
        ElseIf (e.X > 160) And (e.X < 229) And _
           (e.Y > 393) And (e.Y < 417) Then
            If (blnFoundLowerButterfly = False) Then
                MessageBox.Show("You found the lower butterfly!")
                intNumFound += 1      ' Increment number of items found
                blnFoundLowerButterfly = True   ' Set butterfly found
            End If
        End If

        ' Check if all items found
        If (intNumFound = 2) Then
            MessageBox.Show("You found everything!  Game over.")
            Me.Close()  ' Exit
        End If
    End Sub
End Class
```

Timer Control

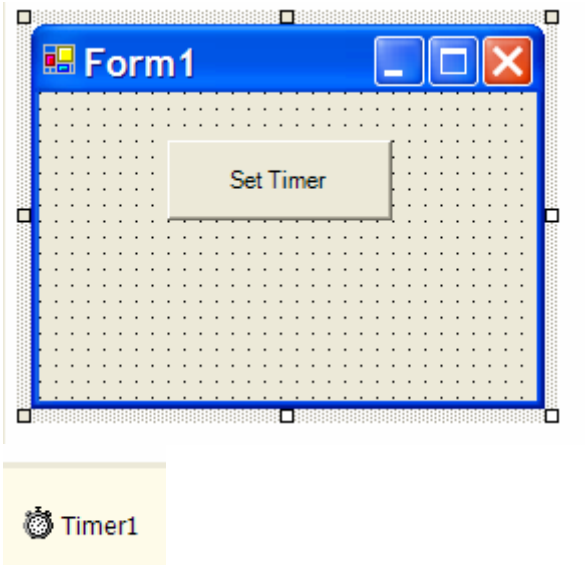
The timer control is used to execute code after some time interval has passed. It is covered in Chapter 8 of the textbook. Unlike all of the other controls we have used so far, it is invisible at runtime. VB.NET places invisible controls in a pane at the bottom of the screen instead of within your form.

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.

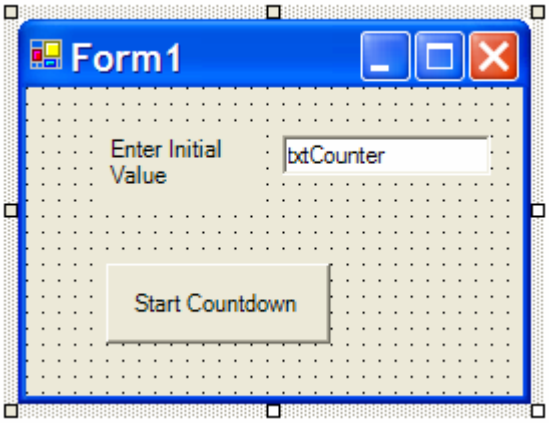
The event that occurs when the timer reaches zero is the Timer1.Tick event. You can add code to this event by double-clicking on the Timer control. The following example initializes the timer and then displays a message after five seconds are up:



```
Private Sub Timer1_Tick(…) Handles Timer1.Tick
    ' Display a message
    MessageBox.Show("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
        MessageBox.Show("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
    MessageBox.Show ("Beep beep!")
    Timer1.Enabled = False   ' Turn timer off
End If
```

Can you figure out what is going on?