Drawing Graphics in C Sharp

http://www.techotopia.com/index.php/Drawing_Graphics_in_C_Sharp#Creating_a_Graphics_Object

The purpose of this chapter of C# Essentials is to provide the reader with knowledge of the basics of graphics drawing in C#. Drawing in C# is achieved using the *Graphics Object*. The Graphics Object takes much of the pain out of graphics drawing by abstracting away all the problems of dealing with different display devices and screens resolutions. The C# programmer merely needs to create a Graphic Object and tell it what and where to draw.

Persistent Graphics

An important point to note before proceeding with is chapter is that simply creating a Graphics Object for a component and then drawing on that component does not create persistent graphics. In fact what will happen is that as soon as the window is minimized or obscured by another window the graphics will be erased.

For this reason, steps need to be taken to ensure that any graphics are persistent. Two mechanisms are available for achieving this. One is to repeatedly perform the drawing in the *Paint()* event handler of the control (which is triggered whenever the component needs to be redrawn), or to perform the drawing on a bitmap image in memory and then transfer that image to the component whenever the *Paint()* event is triggered. We will look at redrawing the graphics in the *Paint()* event in this chapter and Using Bitmaps for Persistent Graphics in C# in the next chapter.

Creating a Graphics Object

The first step in this tutorial is to create a new Visual Studio project called CSharpGraphics. With the new project created select the Form in the design area and click on the lightning bolt at the top of the *Properties* panel to list the events available for the Form. Double click the *Paint* event to display the code editing page.

Graphics Objects are created by calling the *CreateGraphics()* method of the component on which the drawing is to performed. For example, a Graphics Object can be created on our Form called Form1 by calling *CreateGraphics()* method as follows in the Paint() method:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     System.Drawing.Graphics graphicsObj;

     graphicsObj = this.CreateGraphics();
 }
```

Now that we have a Graphic Object we need a Pen with which to draw.

Creating a Pen In C#

A Graphics Object is of little use without a Pen object with which to draw (much as a sheet of paper is no good without a pen or pencil). A Pen object may be quite easily created as follows:

Pen *variable_name* = new Pen (*color*, *width*);

where *variable_name* is the name to be assigned to the Pen object, *color* is the color of the pen and *width* is the width of the lines to be drawn by the pen.

For example, we can create red pen that is 5 pixels wide as follows:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     System.Drawing.Graphics graphicsObj;
```

```
        graphicsObj = this.CreateGraphics();

        Pen myPen = new Pen(System.Drawing.Color.Red, 5);
    }
```

Once a Pen object has been created other properties may be changed. For example, the DashStyle property can be modified to change the style of line (i.e Dash, DashDot, DashDotDot, Dot, Solid or Custom). Properties such as the color and width may similarly be changed after a Pen has been created:

```
    myPen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDotDot;

        myPen.Color = System.Drawing.Color.RoyalBlue;

        myPen.Width = 3;
```

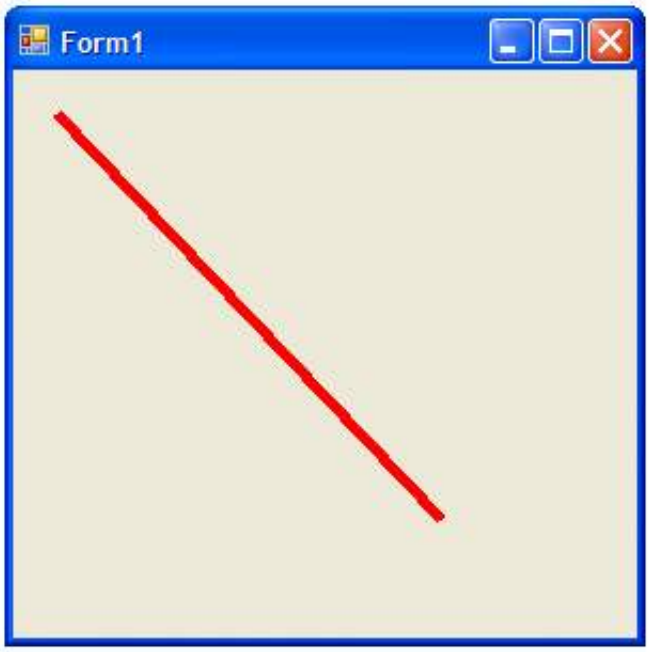
Now that we have a Paint() event handler, a Graphics Object and Pen we can now begin to draw.

Drawing Lines in C#

Lines are drawn in C# using the *DrawLine()* method of the Graphics Object. This method takes a pre-instantiated Pen object and two sets of x and y co-ordinates (the start and end points of the line) as arguments. For example, to draw a line from co-ordinates (20, 20) to (200, 210) on our sample form:

```
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        System.Drawing.Graphics graphicsObj;

        graphicsObj = this.CreateGraphics();

        Pen myPen = new Pen(System.Drawing.Color.Red, 5);

        graphicsObj.DrawLine(myPen, 20, 20, 200, 210);
    }
```

The above code, when compiled and executed will result in the form appearing as follows:



Drawing Squares and Rectangles in C#

For the purposes of drawing rectangles and squares in C# the GraphicsObject provides the *DrawRectangle()* method. There are two ways to use the *DrawRectangle()* method. One is to pass through a *Rectangle* object and Pen and the other is to create an instance of a Rectangle object and pass that through along with the Pen. We will begin by looking at drawing a rectangle without a pre-created Rectangle object. The syntax for this is:

graphicsobj.DrawRectangle(*pen*, *x*, *y*, *width*, *height*);

The alternative is to pass through a Rectangle object in place of the co-ordinates and dimensions. The syntax for creating a Rectangle object in C# is as follows:

Rectangle *rectangleObj* = new Rectangle (*x*, *y*, *width*, *height*);

Once a Rectangle object has been instantiated the syntax to call *DrawRectangle()* is as follows:

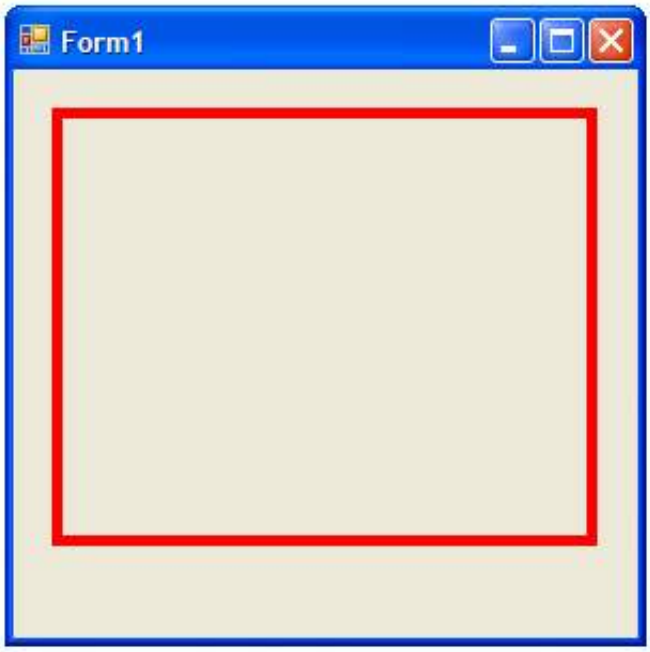
graphicsobj.DrawRectangle(*pen*, *x*, *y*, *rectangleobj*);

The following example creates a Rectangle which is then used as an argument to

DrawRectangle():

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
     System.Drawing.Graphics graphicsObj;

     graphicsObj = this.CreateGraphics();

     Pen myPen = new Pen(System.Drawing.Color.Red, 5);

     Rectangle myRectangle = new Rectangle(20, 20, 250, 200);

     graphicsObj.DrawRectangle(myPen, myRectangle);
}
```

When an application containing the above code is compiled and executed the following graphics will appear in the form:



If multiple rectangles of different shapes need to be drawn it is not necessary to create a new Rectangle object for each call to the *DrawRectangle();* method. Instead the shape of an existing Rectangle object may be altered by calling the *Inflate()* method of the Rectangle class. This method accepts two arguments, the amount by which the width is to be changed and the amount by which the height is to be changed. If a dimension is to be left unchanged 0 should be passed through as the change value.

To reduce a dimension pass through the negative amount by which the dimension is to be changed:

```
Rectangle myRectangle = new Rectangle(20, 20, 250, 200);

myRectangle.Inflate(10, -20); Increase width by 10. Reduce height my
```

Drawing Ellipses and Circles in C#

Ellipses and circles are drawn in C# using the *DrawEllipse()* method of the GraphicsObject class. The size of the shape to be drawn is defined by specifying a rectangle into which the shape must fit. As with the *DrawRectangle()* method, there are two ways to use the *DrawEllipse()* method. One is to pass through a *Rectangle* object and Pen and the other is to create an instance of a Rectangle object and pass that through along with the Pen.

To draw an ellipse without first creating a Rectangle object use the following syntax:

graphicsobj.DrawEllipse(*pen*, *x*, *y*, *width*, *height*);

The alternative is to pass through a Rectangle object in place of the co-ordinates and dimensions. The syntax for creating a Rectangle object in C# is as follows:

Rectangle *rectangleObj* = new Rectangle (*x*, *y*, *width*, *height*);

Once a Rectangle object has been instantiated the syntax to call *DrawRectangle()* is as follows:

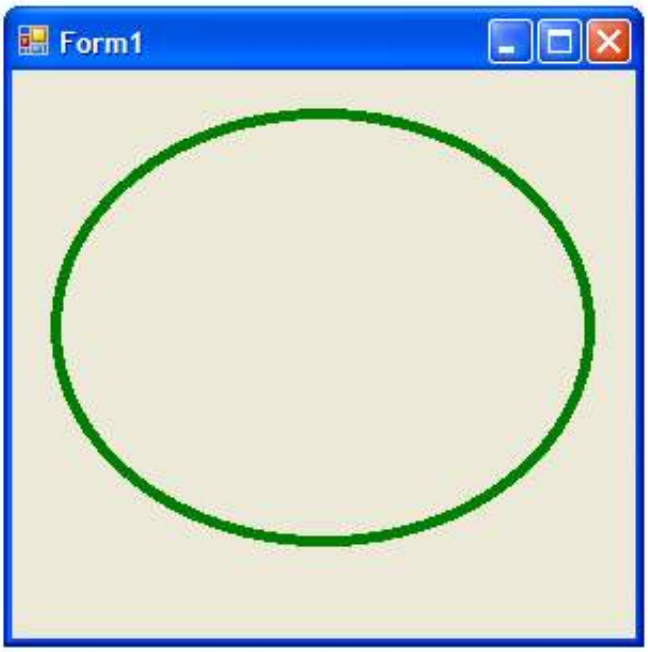
graphicsobj.DrawEllipse(*pen*, *x*, *y*, *rectangleobj*);

The following example creates a Rectangle which is then used as an argument to *DrawEllipse()*:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
     System.Drawing.Graphics graphicsObj;
```

```
            graphicsObj = this.CreateGraphics();

            Pen myPen = new Pen(System.Drawing.Color.Green, 5);
            Rectangle myRectangle = new Rectangle(20, 20, 250, 200);
            graphicsObj.DrawEllipse(myPen, myRectangle);
        }
```

When compiled and executed the above code creates the following graphics output on the form:



Drawing Text with C#

Text is drawn onto a Graphics Object using the *DrawText()* method. The syntax for this method is as follows:

graphicsobj.DrawString(*string*, *font*, *brush*, *x*, *y*);

The *string* argument specifies the text to be drawn. Font defines the font to be used to display the text and requires the creation of a *Font* object. The *brush* object is similar to the *Pen* object used to draw shapes with the exception that it specifies a fill pattern. Finally, the x and y values specify the top left hand corner of the text.

In order to create a Font object a font size, font family and font style may be specified. For example to create a Helvetica, 40 point Italic font:

```
Font myFont = new System.Drawing.Font("Helvetica", 40, FontStyle.Ita
```

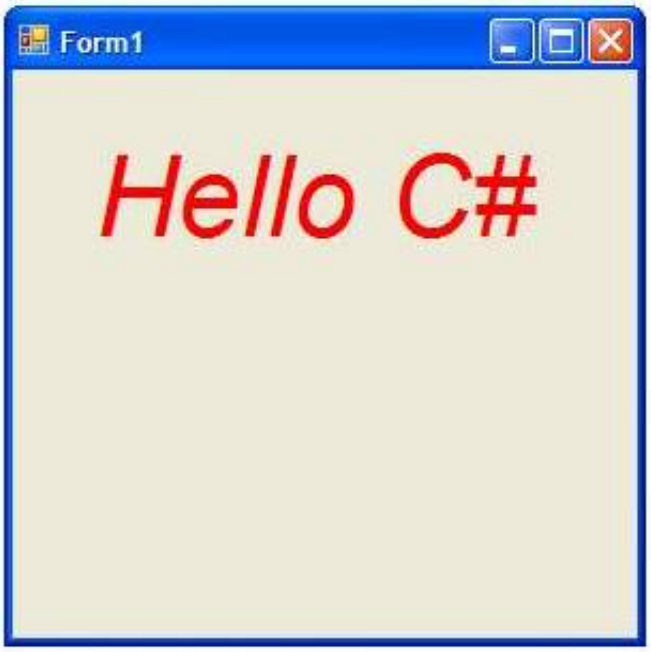
A brush object is created by specifying by calling the appropriate constructor for the brush type and specifying a color:

```
Brush myBrush = new SolidBrush(System.Drawing.Color.Red);
```

Having created the necessary objects we can incorporate these into our example C# application to draw some text:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
     System.Drawing.Graphics graphicsObj;

     graphicsObj = this.CreateGraphics();

     Font myFont = new System.Drawing.Font("Helvetica", 40, FontStyl

     Brush myBrush = new SolidBrush(System.Drawing.Color.Red);

     graphicsObj.DrawString("Hello C#", myFont, myBrush, 30, 30);

 }
```

The above code, when compiled and run, will output the following text onto the form:



Previous	Table of Contents	Next
Building a Toolbar with C Sharp and Visual Studio		Using Bitmaps for Persistent Graphics in C#