Drawing in VB.NET Part 1

What is drawing?

An introduction to drawing seems like a good part of the first couple of articles in my new VB.NET tutorials. Drawing is crucial as part of VB.NET if you want to create custom controls which have a different appearance to the normal ones. Controls which use graphics code to render them most use GDI+ (graphics device interface) which allows for control over a low level of screen printing commands. Painted values have issues however, especially when using GDI+. GDI is an API developed by Microsoft and has been around for a long time. It has always been designed for creating lines that make up the screen but it is incapable of drawing games.

This article could allow you to build a simple game that only uses 2D modelling, but as that is not my forte or interest, I will be showing you how to build simple shapes.

What is a bitmap

First off, you must grasp the concept of a bitmap. A bitmap is a map of bits, i.e. there are bits for each pixel within an image. So for an 8 bit image (i.e. 256 colours as this is the maximum number that 8 bits can represent) a single pixel is represented by 8 bits or eight 1s or 0s. This bit represents the colour. If the pointer that specifies the colour depth is told to represent 24 bit-depth and the user was to only allocate 8 bits per pixel, the image would not display as there will be missing bits (16 missing bits per pixel, so what does it do?). The image would remove all the pixels when it passes the last bit, as it would assume that to be the last pixel.

A bitmap can be represented as a table:

1000 1111	1001 1111	1011 1111	0001 1000
0000 0000	0000 1111	0000 1011	1100 0111
0001 1111	0111 0000	1111 1111	1010 0101
0111 1111	1101 1010	1010 1010	1010 0101

As RGB values cannot have negatives or decimal places, we do not need a sign bit or an exponent.

Declaring variables

So now that we have understood what a bitmap is, let us declare one in VB.NET.

```
Dim OurBitmap As New Bitmap(width, height)
```

The bitmap constructor has multiple overloads, but we will use the simplest one available to us:

```
Dim OurBitmap As New Bitmap(Me.Width, Me.Height)
```

This variable, OurBitmap, has been declared as to be the width and height of the containing form. We can specify any arbitrary number that can represent any size of any object. Painter Pro is built upon the concept of bitmaps.

An alternative constructor for the bitmap class is to provide a filename that will be loaded when the bitmap is declared.

```
Dim OurBitmap As New Bitmap(filename)
```

This will slow loading times of the application so it should only be used if necessary.

Graphics

Graphics is a core part of GDI. A bitmap does nothing on its own and cannot be drawn to unless through graphics. In fact, graphics can draw to any control on the form using the `CreateGraphics()` method which seemingly tries to hide from the code completion tool.

Graphics can be connected to a bitmap:

```
Dim g As Graphics = Graphics.FromImage(bitmap)
```

This specifies the bitmap or control to send commands to so that it can access a variety of different methods. One of those methods is the `DrawLine()` method:

```
g.DrawLine(pen, firstPoint, secondPoint)
```

This method can be represented by:

```
g.DrawLine(Pens.Black, New Point(1, 9), New Point(88, 73))
```

This is useful for drawing a simple line across the bitmap from point ((x) 1, (y) 9) through to the point ((x) 88, (y) 73)) with the specified pen colour as black. The `Pen` may also be changed to have many different attributes including line dash. Amazing? Not quite. The `Pen` lacks on a few points:

- Drawing is slow the majority of the time as it uses GDI+
- `Pen` cannot draw thick lines or fill an object
- `Pen` cannot have gradients or any other kind of filler

For the most of the problems encountered by `Pen`, we use `Brushes`.

`Brushes` is an abstract class which covers all types of brush such as the `SolidBrush` which abstractly implements it. `SolidBrush` allows us to fill a rectangle. `SolidBrush` also is a class similar to `LinearGradientBrush` and `TextureBrush`. All of these brushes can be used to fill objects.

In the next section we are going to cover rectangles as well as performing iterative drawings.