

More on Text in .NET

We've already seen the constructor

```
Font (string Family, float sizeInPoints);
```

There is also

```
Font(string Family, float sizeInPoints, FontStyle fs);
```

You can put in

FontStyle.Regular,
FontStyle.Bold,
FontStyle.Italic,
FontStyle.Bold | FontStyle.Italic

But note that the font sizes are given in point sizes. How is this related to pixels? Answer, one point = 1/72 of the value of the *Graphics* object's property *DpiY* (dots per inch Y). Thus it should come out close to right on the printer, or on the screen.

What if we want to mix text with images? Images are measured in pixels, and text in points. How can we position things correctly? Also, what if we want to know whether the mouse is or isn't over a certain piece of text? The mouse handler gets coordinates *e.X* and *e.Y* in pixels.

You can also construct fonts using pixels, inches, millimeters, or "document units" (1/300 of an inch). These are constants in an "enumeration" *GraphicsUnit*, whose members are

GraphicsUnit.Point
GraphicsUnit.Inch

GraphicsUnit.Millimeter
GraphicsUnit.Document
GraphicsUnit.World

We haven't discussed "world coordinates" yet, but we will below.

The constructor

```
Font("Times New Roman", 12)
```

is equivalent to

```
Font("Times New Roman", 12, GraphicsUnit.Point);
```

To space lines of text down a page, you can use

```
y += font.GetHeight(e); // where Graphics e
```

However, you'll probably find that this leaves too little space between the lines, at least with some common fonts, since apparently the font designer made *external leading* zero. In practice you have to adjust your line spacing empirically (that means, by trial and error).

There is also *MeasureString*, a method of the *Graphics* class like *DrawString*. Both these methods take a font argument. *MeasureString* returns a *Size* object, so you can get both the width and height of your string from the return value of *MeasureString*.

World Coordinates

In the *Graphics* class there is a method *ScaleTransform* that we could use like this:

```
Font font = new Font("Times New Roman",10,FontStyle.Italic);
```

```
int cx = ClientSize.Width;
int cy = ClientSize.Height;
String msg = "Damn the torpedoes!";
SizeF sizef = e.MeasureString(msg, font);
float fScaleHorz = cx / sizef.Width;
float fScaleVert = cy / sizef.Height;
e.ScaleTransform(fScaleHorz, fScaleVert);
e.DrawString(msg,font, new SolidBrush(Color.Red),0,0);
```

Now the text will fill up the window. Actually, there is still some space at the top—that is internal leading, as we studied before. We have now defined “world coordinates”. Say, for example, that `cx` was 100, and `sizef.Width` was 10. After the *ScaleTransform* call, the string will occupy 100 pixels instead of 10. One unit in the x-direction will be 10 old units, because *fScaleHorz* was 10.

When presenting text in a window, it might be very convenient to set up coordinates in printers points. You can use *ScaleTransform* to do that. The main reason to do so is that then, the same code can be used to handle a *Paint* event and to print (on a printer).