

Re: Ignored advice, am now in serious poo on graphics

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-11/msg03873.html>

- *From:* Peter Duniho <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 28 Nov 2007 11:48:09 -0800
-

On 2007-11-28 04:47:26 -0800, "Peter Webb" <webbfamily@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> said:

[...]

Through the debugger, I noticed that all the graphics commands executed before the Form actually fully loaded. So I moved the routine to draw the initial graphic into `pictureBox1_Paint` method. I cheated and put a condition around it so it only fired off once at the start. No difference. Then I put in a `thread.sleep(1000)`, after the draw in the Paint method, and noticed a strange thing – the image was shown for 1000 milliseconds, then disappears. If I do the same thing explicitly on a button click, it doesn't disappear.

I am now in a bit of a mess. My only real problem was that the initial graphic in the `pictureBox` would display on a button click, but not on simply loading `Form1`. It looked like a timing problem, but neither `DoEvents` or `Thread.Sleep` solves it. Is there some simple way of solving my original problem? Can anybody tell me what is going on?

It's hard to say without seeing the code. From your description, you seem to be now actually handling the Paint event. But you also write that you only execute your code once. You need to always be prepared to handle the Paint event, and always draw everything when it's raised.

But without knowing what your code looks like, it's difficult to say for sure what you've done wrong.

For what it's worth, I don't really think a `PictureBox` is an appropriate control here. If you're doing your own drawing, there's nothing in a `PictureBox` control that is useful to you. Alternatively, if you want to use a `PictureBox` then you don't want to handle that `PictureBox`'s Paint event. Instead you want to generate a `Bitmap` instance that you can assign to the `PictureBox` as its `Image`, and let the `PictureBox` deal with the redrawing issues.

The latter may actually be closer to what you're trying to do anyway. After all, you seem set on only drawing things once. But it's hard to know for sure without any real code examples from you.

If you do choose to go the "Windows-approved" route, handling the paint message Windows sends the control, here's the basic idea:

* Create your own custom control. Unless you need some additional behavior, other than the basic paint event handling, provided by some specific control type your custom control should just derive from `Control`.

Re: Ignored advice, am now in serious poo on graphics

* Override the OnPaint() method of Control. In this override method, put all of your drawing code. Make sure that your drawing code can always draw. Depending on the architecture of your program, this could mean nothing extra or it could mean that you need to synchronize your data structures so that another thread generating data to be drawn isn't accessing the data at the same time your main thread is using it to draw to the screen.

* Any time that the data used to draw to the screen changes in a way that would affect how the drawn data looks on the screen, call your custom control's Invalidate() method. This will inform Windows that the control needs to be redrawn to take into account the new data, and will cause a WM_PAINT message to be sent to your control, resulting in your OnPaint() method being called so it can draw.

For extra credit, you can do a couple of other things:

* When data changes, keep track of what area on the screen is actually affected, and use that area as a parameter to the Invalidate() method so that you don't waste time drawing things that haven't changed. Your own code will still act like it's drawing those things, but the parts that haven't actually changed will be "clipped" out of your drawing, improving performance and reducing flicker (the flicker happens because when you invalidate the control, the first thing Windows does as part of the redraw is to erase what was there so that you are starting with a clean slate...the repeated erasing and redrawing causes the flicker)

* In your custom control's constructor, set the DoubleBuffered property to true. This will eliminate flicker by causing all of your drawing to take place into an off-screen buffer, erasing and all. Then that buffer is copied automatically to the screen by .NET when you're done.

Graphical rendering of data can actually be fairly complicated. It would take a lot more than a single newsgroup post to address all of the complexities of graphics under Windows. But the above really is the basic starting point, and is all the detail one needs to know for a great many kinds of applications. More importantly, no matter how complicated the application, they all need to use the same foundation described above.

Deviate from that foundation at your own peril. :)

Pete

.