Animation and Effects
CIS 399-005 Notes

March 25, 2009

1 Full Screen

To make a form display in fullscreen, set its WindowState and BorderStyle properties as below.

```
Form f = ...
f.WindowState = FormWindowState.Maximized;
f.FormBorderStyle = FormBorderStyle.None;
```

You can do this either in code explicitly or using the form designer.

2 Drawing on Forms

You can draw on forms just like custom controls. Override the OnPaint method and provide drawing code.

```
protected override void OnPaint(PaintEventArgs e)
{
  base.OnPaint(e);

  var g = e.Graphics;
  g.FillEllipse(Brushes.OliveDrab,
                new Rectangle(10, 20, 150, 80));
}
```

When the screen needs to be redrawn the form will receive a Paint message. The above override is called because the Form base-class has already installed a handler which calls the OnPaint virtual method.

3 Quiting nicely

Because we've created a full-screen window, we can't quit the running program by using the border's close button. By handling the KeyDown event, we can respond to key presses and quit the application when a user pushes the "q" key.

```
private void AnimationDemo_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyData == Keys.Q)
    {
        Application.Exit();
    }
}
```

It's worth pausing to observe that we've used two distinct techniques to alter the behavior of our form. We used overriding to change OnPaint but handled the KeyDown message directly. Both techniques are fine. However, for your own sanity, you should use one consistently in a given project.

4 Displaying Text

Although the user can quit, he has no visual indication of this. We can updated OnPaint to rectify the situation.

```
...
g.DrawString("Press Q to Quit",
             new Font("Comic Sans MS", 15, FontStyle.Regular),
             Brushes.SeaGreen,
             200, 50);
```

5 Drawing an Image

Let's try to display an image.

First find a bitmapped image, such as a jpg. Add this to your project using Project→Add Existing.... The image file will now appear in the Solution Explorer. You can adjust the image's properties; set "Copy to Output Directory" to "Copy if Newer." The jpg will be copied into the same directory as your executable, and it will be easy to find.

Actually displaying the image is easy. Add the following code to OnPaint.

```
...
Image monkeyPhoto = Image.FromFile(Application.StartupPath + "\\monkey.jpg");
g.DrawImage(monkeyPhoto, 80, 100);
```

If your image isn't precisely the size you want, you can resize it using an overload of DrawImage.

```
g.DrawImage(monkeyPhoto, 80, 200,
            monkeyPhoto.Width/10, monkeyPhoto.Height/10);
```

6 Moving the Image

It's now easy to make the image move, using key presses and what we've seen so far. Add a private member, MonkeyY, to track the current image's current height and make the corresponding changes to OnPaint.

At this point we can adjust MonkeyY in the KeyDown handler. As with custom controls, we must call Invalidate to force the screen to redraw.

```
private void AnimationDemo_KeyDown(object sender, KeyEventArgs e)
{
    switch (e.KeyData)
    {
        case Keys.Q:
            Application.Exit();
            break;

        case Keys.Up:
            MonkeyY--;
            break;

        case Keys.Down:
            MonkeyY++;
            break;

        default:
            break;
    }

    this.Invalidate();
}
```

7 Double Buffering

While the above code works, there's a flickering effect when the image moves, and the result is pretty ugly. To fix, set the form's DoubleBuffered property to true.

Double buffered forms do all their drawing off-screen, then update the onscreen image all at once.

8 Adding transparency

Using MS Paint (or Gimp or Photoshop) color the background of your image in a single solid color. I'll use red, in particular all the way red (255,0,0).

We can then use another override of DrawImage to draw image. It takes an additional argument of type System.Drawing.Imaging.ImageAttribute, which controls transparency and several other image properties including gamma-correction.

We modify OnPaint as follows.

```
ImageAttributes attr = new ImageAttributes();
attr.SetColorKey(Color.Red, Color.Red);

Image monkeyPhoto = Image.FromFile(Application.StartupPath +
                                   "\\monkey-red.bmp");
```

```
var r = new Rectangle(80, MonkeyY,
                      monkeyPhoto.Width / 10, monkeyPhoto.Height / 10),
g.DrawImage(monkeyPhoto, r,
            // Area of source image to use---use it all
            0, 0, monkeyPhoto.Width, monkeyPhoto.Height,
            // Specify the units of measurement
            GraphicsUnit.Pixel,
            // Attribute controlling transparency
            attr );
```

9 Animating the monkey

We animate the image, by making it move at regular intervals. The following code (all part of the AnimationDemo class) creates a new Timer object that fires an event at regular intervals. The OnTick method handles this event by altering the images position an invalidating the screen.

```
private int MonkeyY = 100;
private int MonkeyX = 80;
private Timer myTimer;

public AnimationDemo()
{
    InitializeComponent();

    myTimer = new Timer();
    myTimer.Interval = 40; // tick every 40 ms (25 fps)
    myTimer.Tick += OnTick;
    myTimer.Enabled = true;
}

private void OnTick(object Sender, EventArgs e)
{
    MonkeyX += 3;
    this.Invalidate();
}
```

We can drop the Invalidate call from the KeyDown handler. Redrawing occurs periodically, so there's no reason for an extra redraw.