Bitmaps, Icons and Cursors

Background

- Bitmaps are easy in .NET.
- Two kinds of graphics:
  1. Vector
  2. Bitmaps
- Vectors are scalable without loss
- Bitmaps are not scalable without loss or artifacts.

Colors

- Several models, e.g. RGB, CYMK etc.
- Color depth is determined the number of bits per pixel.
- Windows uses 24 bits.
- An *alpha channel* can be used to specify transparency.

Bitmap Files

- The .NET FCL supports reading and writing of many standard file formats.
- This makes working with bitmaps much easier than say MFC.
- Compression is used with some formats that result in some loss of resolution.

Common File Formats

FIle Format	
.bmp	Standard Windows bitmap (not compressed).
.gif	Graphics Interchange Format (lossless compressions); popular for Web applications.
.jpg	Joint Photographic Experts Group (compressed with loss of detail); extremely popular for all types of applications.
.png	Portable Network Graphics (lossless compression) Newer standard for Web applications.
.tiff	Tag Image File Format (various compression algorithms); older format that is still used.
.exif	Exchangeable Image File (uses JPEG compression); popular for digital cameras.

The Bitmap Class

- *Bitmap* is derived from *Image*.
- A Bitmap can be created directly from a file with no file I/O required.
- Bitmaps can be created easily by a program and then written to the file format of your choice.

Bitmap Constructors

Bitmao Constructor	*Description*
Bitmap(Image original)	Create a new bitmap from an existing Image object.
Bitmap(string filename)	Creates a new bitmap from a file.
Bitmap(int width, int height)	Creates a new bitmap that is empty (all pixels are zero) of the specified size.

Drawing a Bitmap

```
Bitmap bm = new Bitmap("myfile.jpg");
g.DrawImage(bm, 0, 0);
```

- It’s really this simple!
- There are some details we will need to discuss such as the exact size of the bitmap that is displayed.
- Is it always the pixel dimension of the file?

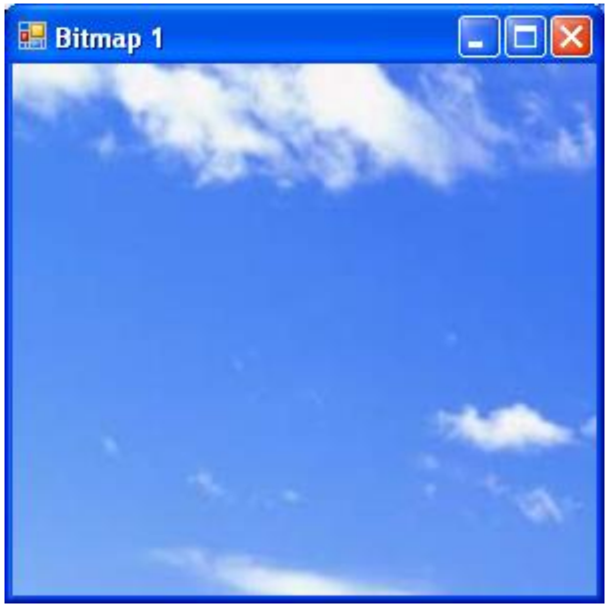
Bitmap1 Example

```
Bitmap1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Bitmap1
{
    public partial class Form1 : Form
    {
```

Bitmap1 Example

```
 private Bitmap bm;
        public Form1()
        {
            InitializeComponent();
            bm = new Bitmap(@"..\..\bliss.bmp");
        }
        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.DrawImage(bm, 0, 0);
        }
    }
}
```

Output



The bitmap is 800 x 600 and we don't see the entire image here.

Resizing Bitmaps

- The next slide shows the properties for the image.
- The DPI (dots per inch) value is used to scale the display.
- Windows default is 96 DPI.
- The image is scaled by 96/72 and so it displays larger than the actual one to one pixel size.

Bitmap Properties

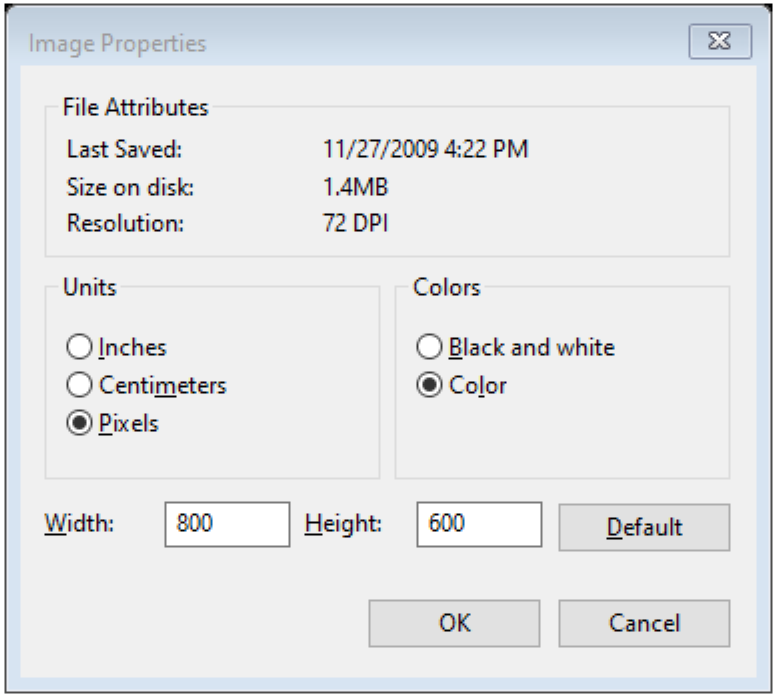


Image Properties

File Attributes

Last Saved:	11/27/2009 4:22 PM
Size on disk:	1.4MB
Resolution:	72 DPI

Units: Inches [✓ Pixels] Centimeters

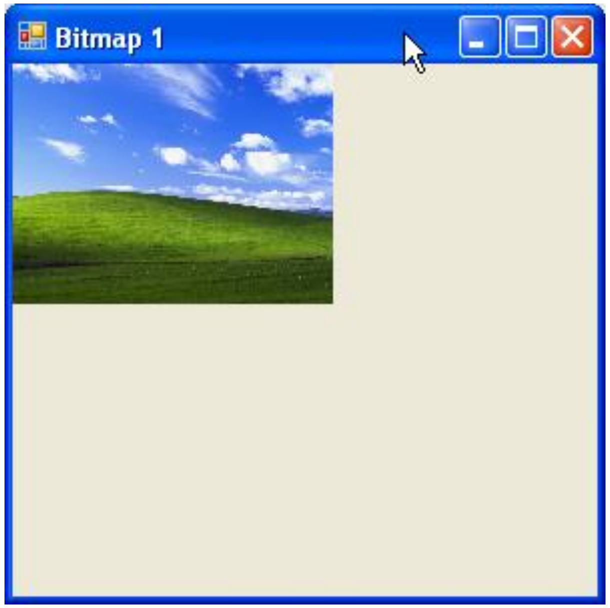
Colors: Black and white [✓ Color]

Width: 800 Height: 600

Setting the Size Manually

```
g.DrawImage(bm, 0, 0, 160, 120);
```

The result



Note – the aspect ratio is not maintained automatically.

Bitmap Properties

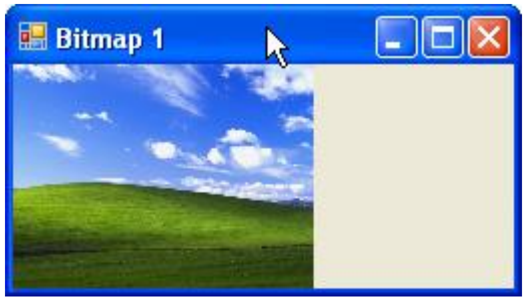
Bitmap Properties	*Description*
Size Size	Size in pixels (width and height)
int Width	Width in pixels
int Height	Height in pixels
float HorizontalResolution	Horizontal resolution in DPI
float VerticalResolution	Vertical resolution in DPI

```
g.DrawImage(bm, 0, 0, bm.Width, bm.Height);
// Draws actual size in pixels.
```

Displays Bitmap to Fit the Client Area

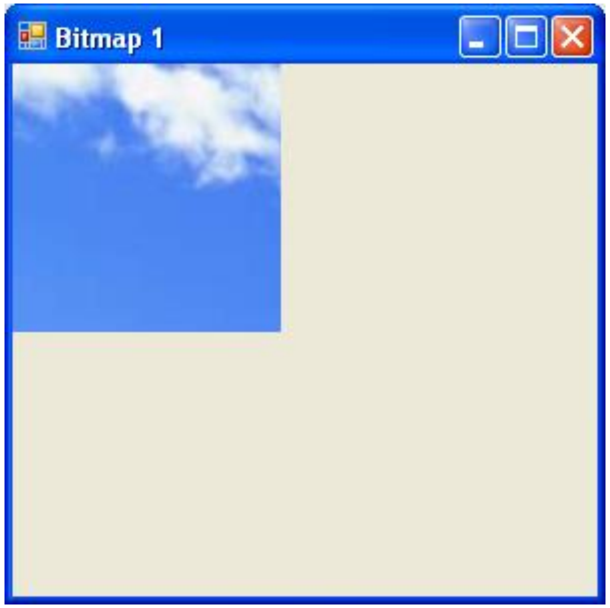
```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    SizeF cs = this.ClientSize;
    float ratio = Math.Min(cs.Height/bm.Height,
        cs.Width/bm.Width);
    g.DrawImage(bm, 0, 0, bm.Width*ratio,
        bm.Height*ratio);
}
```

Result



Displaying Part of a Bitmap

```
g.DrawImage(bm, 0, 0, new Rectangle(0,0,100,100),
    GraphicsUnit.Pixel);
```



Embedding Image Resources

- Drawbacks to image files:
  1. You need to deploy the image file with the application.
  2. If the image file is deleted the application will fail.
  3. The end user can modify the application by editing the image file.
- Embedding solves these problems.

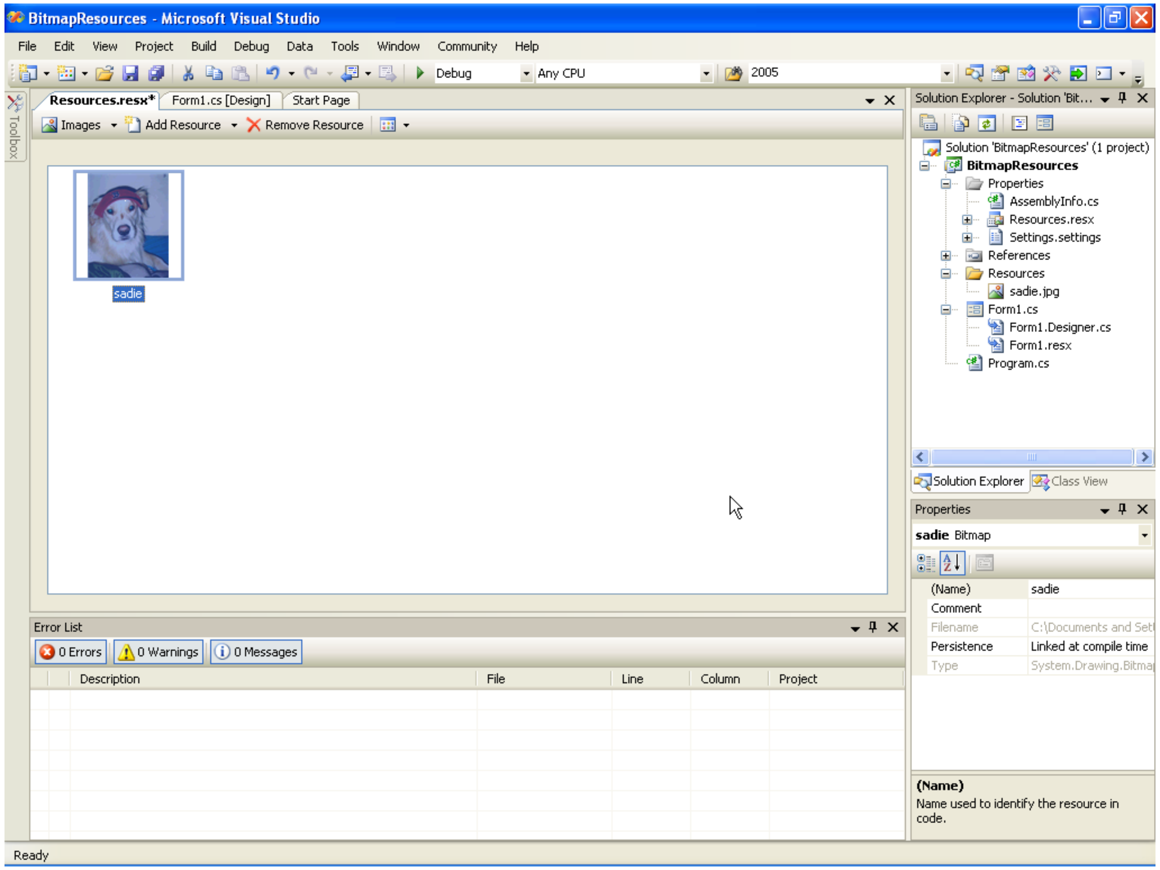
Use the Resource Designer

- The *Resource Designer* allows you to add the following resource types to the assembly:
  1. Strings
  2. Images
  3. Icons
  4. Audio
  5. Files
  6. Other types
- *Resources.resx* and *Resources.Desginer.cs are key files used*.

Using the Resource Designer

- Double click on the *Resources.resx* file in the *Properties* folder.
- Select *Images* from the leftmost tab and then select *Add Exisiting File* from the drop down list on the *Add Resource* tab.
- Navigate to the file you want and select it. I added the file *sadie.jpg* and the result is shown in in the following slide.
- As you can see the file has also been added to the project in the *Resources* folder. You can rename your images from the default which is the name of the image file without the file extension.

Resource Designer



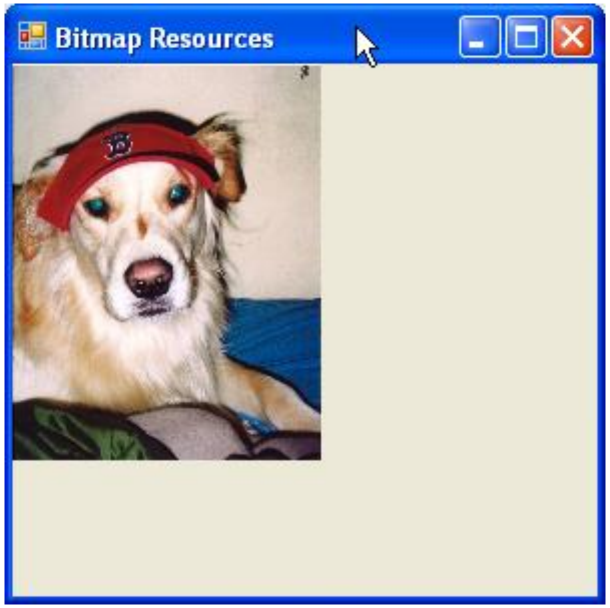
(Name)	sadie
Comment	
Filename	C:\Documents and Set
Persistence	Linked at compile time
Type	System.Drawing.Bitma

(Name)
Name used to identify the resource in code.

Display the Bitmap

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Bitmap bm = Properties.Resources.sadie;
    g.DrawImage(bm, 0, 0);
    bm.Dispose();
}
```

Sadie



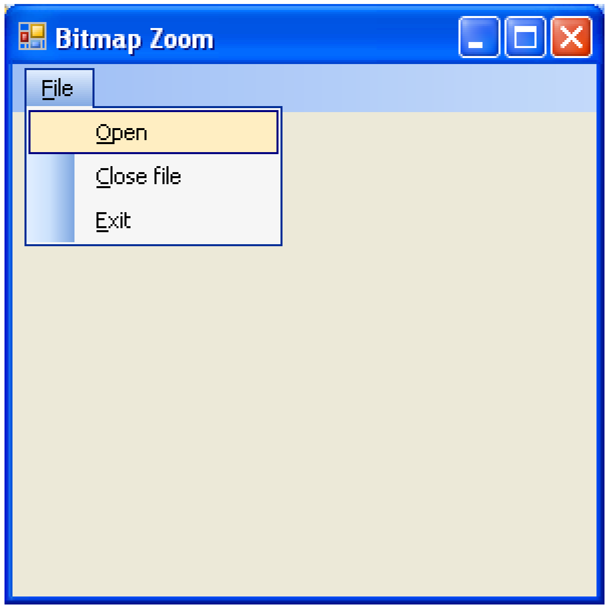
Hidden Designer Code

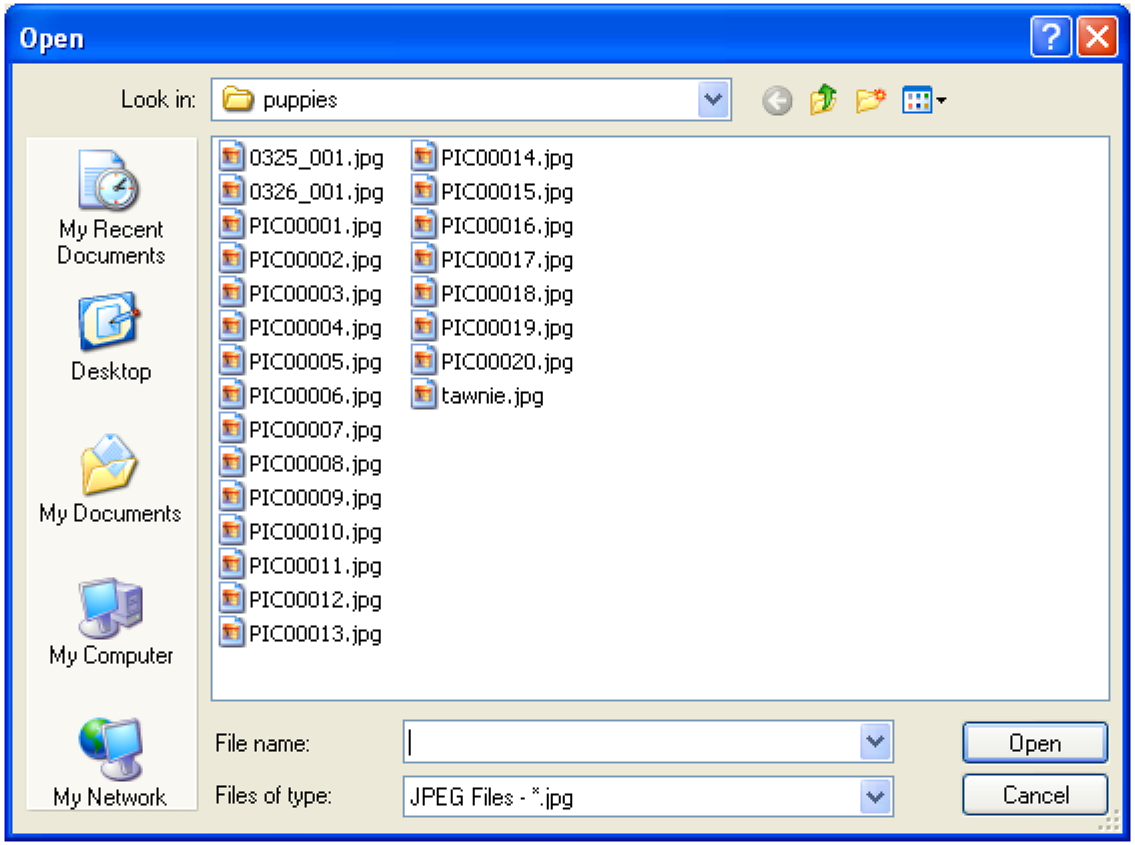
```
internal static System.Drawing.Bitmap sadie {
    get {
        object obj = ResourceManager.GetObject("sadie"),
            resourceCulture);
        return ((System.Drawing.Bitmap)(obj));
    }
}
```

Note the cast

Zoom Example

- This is a clever example demonstrating a number of techniques in addition to just displaying a bitmap.
- One is the use of a *rubber band box*.
- The slides that follow show the application at work.

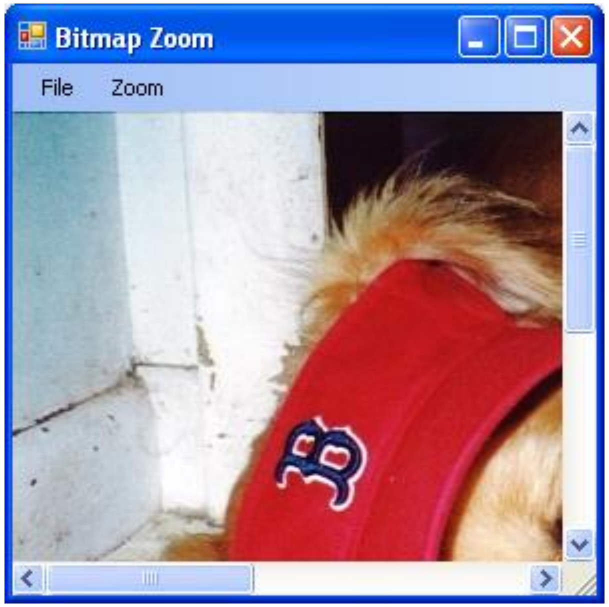

Bitmap Zoom
File
Open
Close file
Exit

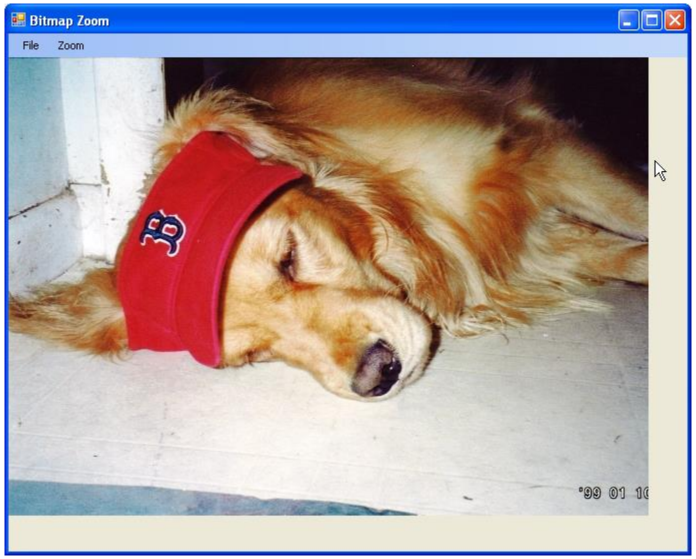


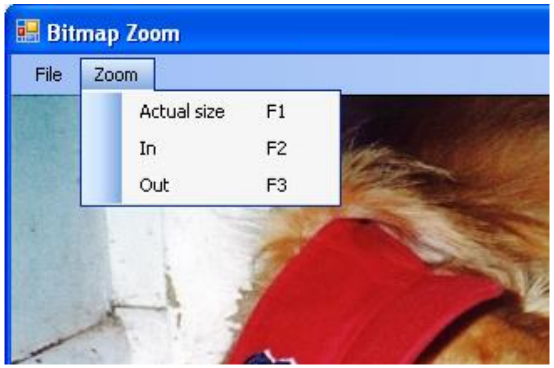
0325_001.jpg
0326_001.jpg
PIC00001.jpg
PIC00002.jpg
PIC00003.jpg
PIC00004.jpg
PIC00005.jpg
PIC00006.jpg
PIC00007.jpg
PIC00008.jpg
PIC00009.jpg
PIC00010.jpg
PIC00011.jpg
PIC00012.jpg
PIC00013.jpg
PIC00014.jpg
PIC00015.jpg
PIC00016.jpg
PIC00017.jpg
PIC00018.jpg
PIC00019.jpg
PIC00020.jpg
tawnie.jpg

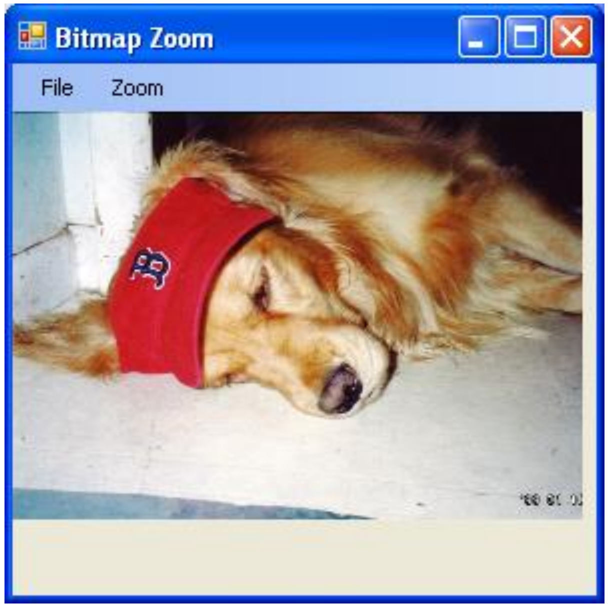
File name:
Files of type: JPEG Files - *.jpg

Zooming

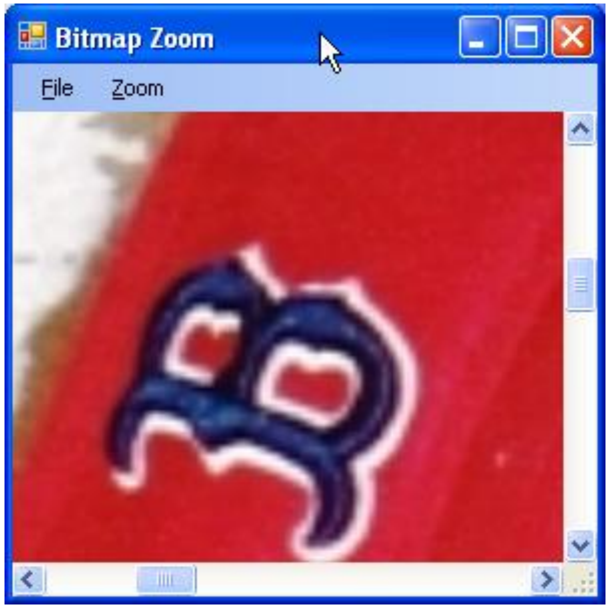




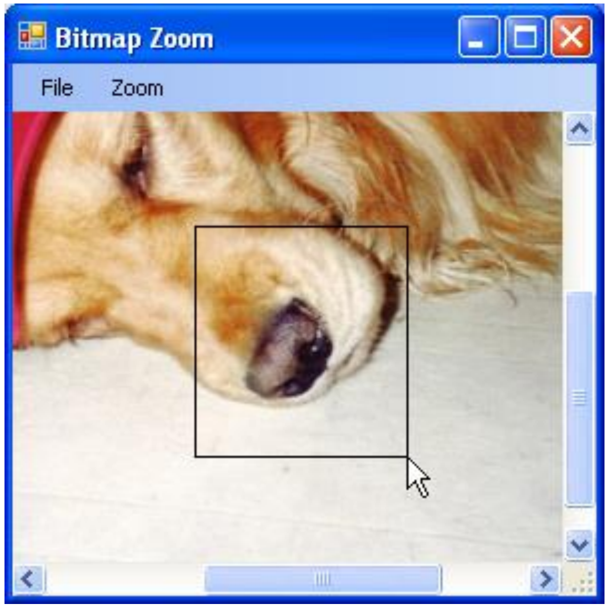

Bitmap Zoom
File
Zoom
Actual size F1
In F2
Out F3

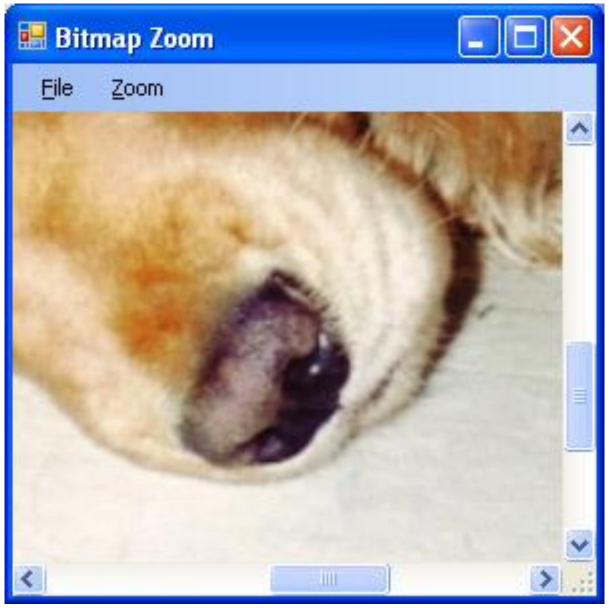

Bitmap Zoom
File
Zoom

Tight Zooming



Rubber Band Box





Zoom Code

- Demo in VS 2019.

Eliminating Flicker

- *Double buffering* is the secret.
- Set the *DoubleBuffered* property to *true*.
- Unfortunately this is a *protected* property and can’t be used with controls unless you derive from them.
- This presents a problem if you use a panel for your output.

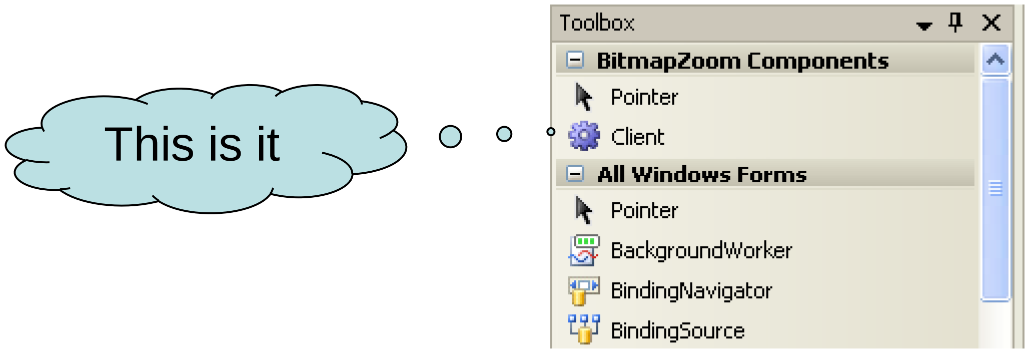
Solution

- Derive from the *Panel* class and set the property in the constructor.

```
public class Client : Panel
{
    public Client()
    {
        DoubleBuffered = true;
    }
}
```

Designer Feature

- If you build your project after creating the derived class the new control will be added to the toolbox automatically.
- You can then use it like any other control.



Drawing on a Bitmap

- We can create a new empty bitmap with:

```
Bitmap bm = new bm(160, 120);
```

- All pixels are initially black (0,0,0).
- We can get a *Graphics* object to draw on the bitmap:

```
Graphics g = Graphics.FromImage(bm);
```

- Draw on the bitmap as you would the screen.

Drawing on a Bitmap

Bitmap Draw - Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace BitmapDraw
{
    public partial class Form1 : Form
    {
        Bitmap bm = null;
```

Drawing on a Bitmap

```
public Form1()
{
    InitializeComponent();
}
private void Form1_MouseDown(object sender,
      MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        if (bm == null) bm = new Bitmap(160, 120);
        Graphics g = Graphics.FromImage(bm);
        g.Clear(Color.White);
```

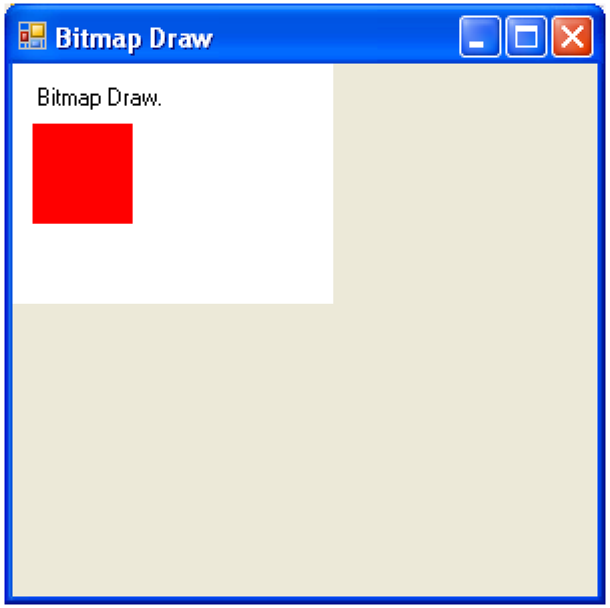
Drawing on a Bitmap

```
        g.DrawString("Bitmap Draw.", Font, Brushes.Black,
                  10, 10);
               g.FillRectangle(Brushes.Red, 10, 30, 50,
                  50);
               g.Dispose();
               Invalidate();
           }
           else
           {
               bm.Dispose();
               bm = null;
               Invalidate();
           }
       }
```

Drawing on a Bitmap

```
        protected override void OnPaint(PaintEventArgs e)
        {
            if (bm!=null)
                e.Graphics.DrawImage(bm, 0, 0, bm.Width,
                      bm.Height);
        }
    }
}
```

Output



Saving a Bitmap to a File

- This is a piece of cake.
- We need to specify the file name and the type of the file.
- Careful, the file type is NOT determined by the file extension.

```
bm.Save("filename.jpg", ImageFormat.Jpeg);
```

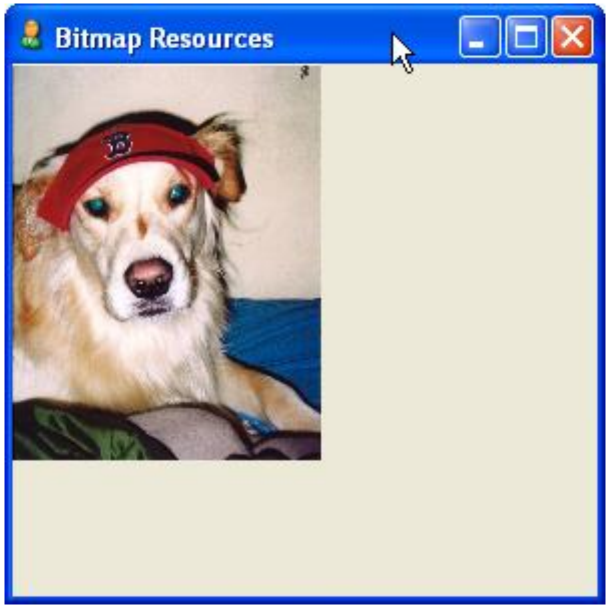
Changing the Icon

- Add an icon to the resources.
- Optionally create an icon (icons are discussed shortly).
- Add this line of code to the constructor:

```
Icon = Properties.Resources.user;
```

- user is the icon name
- You can also change using the form Icon property.

New Icon



See this?

Icons – Two Types

1. An icon for the application itself. This is associated with the *.exe* file and is displayed when you view the folder where the file is located using the Windows Explorer. A large size icon appears if you view the folder with the *Icon* view, otherwise a small version is used. If you place a shortcut or the application itself on the desktop you will also see this icon. To change the icon open the properties window for the project.
2. An icon for the form itself. This icon appears in the upper left hand corner of the form and is used as a button to open the *system* menu. It also appears when you minimize the form to the task bar.

Sizes

- Standard icons are either 16 x 16 or 32 x 32 pixels in 16 colors.
- If a 32 x 32 icon is the only icon available then it will be scaled to 16 x 16 with sometimes marginal results.
- The reverse is also true.

Cursors

- I already showed an example of changing the cursor.
- You can create your own cursors.
- A cursor has a *hot spot* that you need to specify.
- VS has a nice cursor editor.