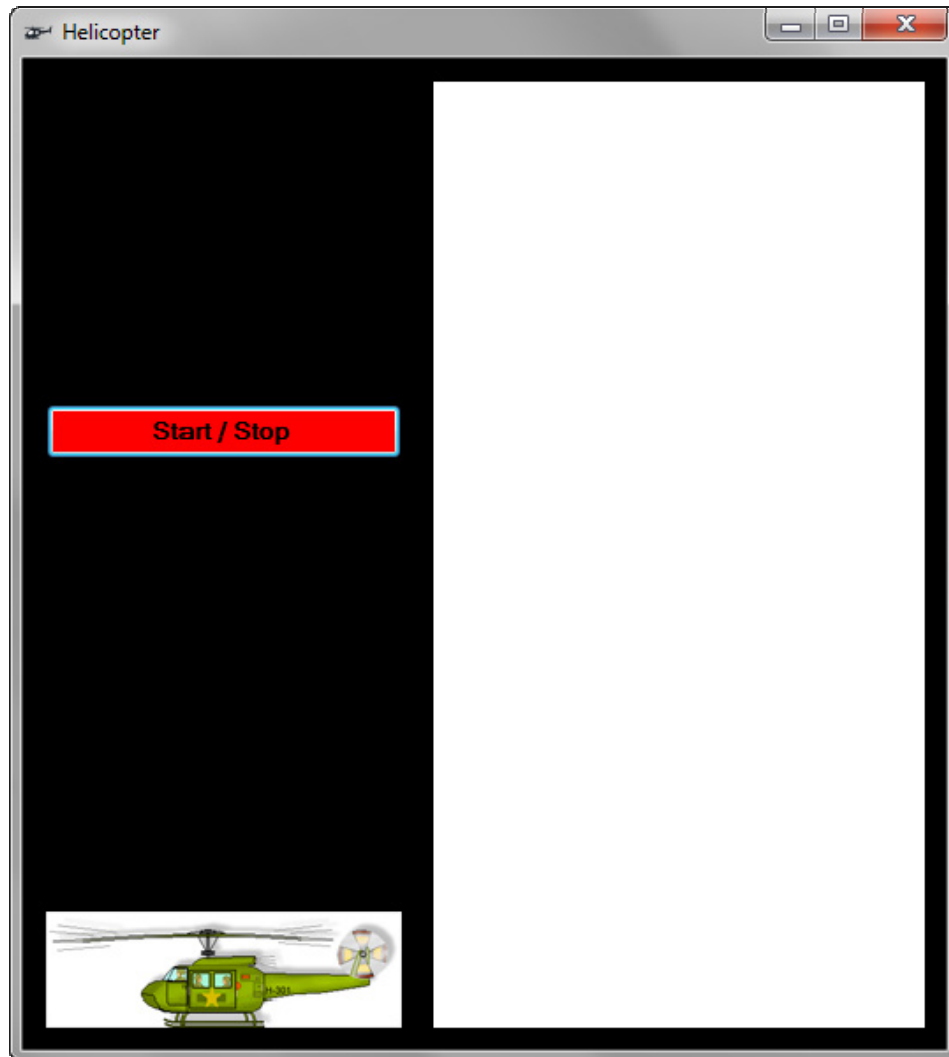


Project 01 – Simple Animation using Graphics

Start Visual C# Express and start a new project.

- Put a panel (**pnlDisplay**) on the form - make it fairly tall with a white background color.
- Put a small picture box (**picHelicopter**) on the form. Set its **Image** property to the helicopter picture.
- Place a timer control (**tmrHover**) on the form. Use an Interval property of 50.
- Place a button (**btnStartStop**) on the form for starting and stopping the timer.

Try to make it look something like this:




Define the needed graphics object in the **general declarations** area. Also include a variable (**imageY**) to keep track of the vertical position of the image:

```
Graphics myGraphics;  
int imageY;
```

And create the object in the **Form1_Load** event method:

```
private void Form1_Load(object sender, EventArgs e)  
{  
    myGraphics = pnlDisplay.CreateGraphics();  
}
```

It's a good habit to Dispose of your graphics to free up memory once your program closes.

Click on the events  icon in the properties window. Double click on the white space beside the event **FormClosing**.

Dispose of the graphics object in the **Form1_FormClosing** event method:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)  
{  
    myGraphics.Dispose();  
}
```

Use **btnStartStop_Click** to toggle the timer and initialize the position (**imageY**) of **picHelicopter.Image** at the top of the panel control: Using the exclamation mark means NOT which will flip the timer on or off depending the current state.

```
private void btnStartStop_Click(object sender, EventArgs e)  
{  
    tmrHover.Enabled = !(tmrHover.Enabled);  
    imageY = 0;  
}
```

Now, move the image in the **tmrHover_Tick** event:

```
private void tmrHover_Tick(object sender, EventArgs e)
{
    int imageX = 10;
    int imageW = 100;
    int imageH = 32;

    myGraphics.Clear(pnlDisplay.BackColor);
    imageY = imageY + pnlDisplay.Height / 40;
    myGraphics.DrawImage(picHelicopter.Image, imageX, imageY,
        imageW, imageH);
}
```

In this event, the image width (**imageW**) and height (**imageH**) are given values (I chose half my actual picture size), as is the horizontal location (**imageX**). Then, the graphics object is cleared to erase the previous image. The vertical position of the image (**imageY**) is increased by 1/40th of the panel height each time the event is executed. The picture box image is moving down.

- Run the project.
- Click the button to start the timer.
- Watch the image drop.
- Notice the image is scaled to fit the area defined by the DrawImage method.

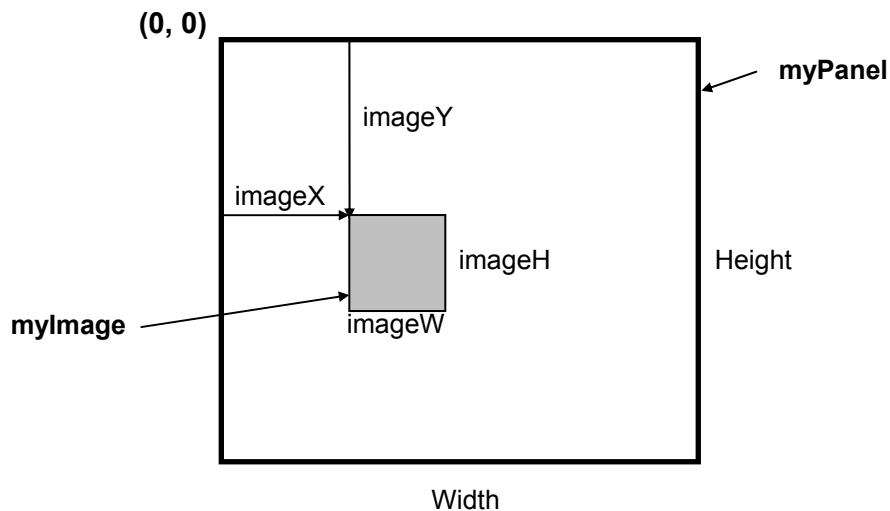
How long does it take the image to reach the bottom?

What happens when it reaches the bottom? It just keeps on going down through the panel, through the form and out through the bottom of your computer monitor to who knows where! We need to be able to detect this disappearance and do something about it.

Image Disappearance

When images are moving in a panel, we need to know when they move out of the panel across a border. Such information is often needed in video type games. When an **image disappearance** happens, we can either ignore that image or perhaps make it “scroll” around to other side of the panel control. How do we decide if an image has disappeared? It’s basically a case of comparing various positions and dimensions.

We need to detect whether a image has completely moved across one of four panel borders (top, bottom, left, right). Each of these detections can be developed using this diagram of a picture box image (**myImage**) within a panel (**myPanel**):



Notice the image is located at (**imageX**, **imageY**), is **imageW** pixels wide and **imageH** pixels high.

If the image is moving down, it completely crosses the panel bottom border when its top (**imageY**) is lower than the bottom border. The bottom of the panel is **myPanel.Height**. C# code for a bottom border disappearance is:

```
if (imageY > myPanel.Height)
{
    [C# code for bottom border disappearance]
}
```

If the image is moving up, the panel top border is completely crossed when the bottom of the image (**imageY + imageH**) becomes less than 0. In C#, this is detected with:

```
if ((imageY + imageH) < 0)
{
    [C# code for top border disappearance]
}
```

If the control is moving to the left, the panel left border is completely crossed when image right side (**imageX + imageW**) becomes less than 0. In C#, this is detected with:

```
if ((imageX + imageW) < 0)
{
    [C# code for left border disappearance]
}
```

If the image is moving to the right, it completely crosses the panel right border when its left side (**imageX**) passes the border. The right side of the panel is **myPanel.Width**. C# code for a right border disappearance is:

```
if (imageX > myPanel.Width)
{
    [C# code for right border disappearance]
}
```

Border Crossings

What if we want the image to bounce back up when it reaches the bottom border? This is another common animation task - detecting the initiation of **border crossings**. Such crossings are used to change the direction of moving images, that is, make them bounce. How do we detect border crossings?

The same diagram used for image disappearances can be used here. Checking to see if an image has crossed a panel border is like checking for image disappearance, except the image has not moved quite as far. For top and bottom checks, the image movement is less by an amount equal to its height value (imageH). For left and right checks, the control movement is less by an amount equal to its width value (imageW). Look back at that diagram and you should see these code segments accomplish the respective border crossing directions:

```
if (imageY < 0)
{
    [C# code for top border crossing]
}

if ((imageY + imageH) > myPanel.Height)
{
    [C# code for bottom border crossing]
}

if (imageX < 0)
{
    [C# code for left border crossing]
}

if ((imageX + imageW) > myPanel.Width)
{
    [C# code for right border crossing]
}
```

Let's modify the falling image example to have it bounce when it reaches the bottom of the panel. Declare an integer variable **imageDir** in the **general declarations** area:

```
int imageDir;
```

imageDir is used to indicate which way the image is moving. When imageDir is 1, the image is moving down (imageY is increasing). When imageDir is -1, the image is moving up (imageY is decreasing).

Change the **btnStartStop_Click** event to (new line is shaded):

```
private void btnStartStop_Click(object sender, EventArgs e)
{
    tmrHover.Enabled = !(tmrHover.Enabled);
    imageY = 0;
    imageDir = 1;
}
```

We added a single line to initialize imageDir to 1 (moving down).

Change the **tmrHover_Tick** event to this (again, changed and/or new lines are shaded):

```
private void tmrHover_Tick(object sender, EventArgs e)
{
    int imageX = 10;
    int imageW = 100;
    int imageH = 32;

    myGraphics.Clear(pnlDisplay.BackColor);
    imageY = imageY + imageDir * pnlDisplay.Height / 40;
    myGraphics.DrawImage(picHelicopter.Image, imageX, imageY,
        imageW, imageH);
    if (imageY + imageH > pnlDisplay.Height)
    {
        imageY = pnlDisplay.Height - imageH;
        imageDir = -1;
    }
}
```

We modified the calculation of imageY to account for the imageDir variable. Notice how it is used to impart the proper direction to the image motion (down when imageDir is 1, up when imageDir is -1). We have also replaced the code in the existing if structure for a bottom border crossing. Notice when a crossing is detected, the image is repositioned (by resetting imageY) at the bottom of the panel (**pnlDisplay.Height - imageH**) and imageDir is set to -1 (direction is changed so the image will start moving up). Run the project. Now when the image reaches the bottom of the panel, it reverses direction and heads back up. We've made the image bounce! But, once it reaches the top, it's gone again!

Add top border crossing detection, so the **tmrHover_Tick** event is now:

```
private void tmrHover_Tick(object sender, EventArgs e)
{
    int imageX = 10;
    int imageW = 100;
    int imageH = 32;

    myGraphics.Clear(pnlDisplay.BackColor);
    imageY = imageY + imageDir * pnlDisplay.Height / 40;
    myGraphics.DrawImage(picHelicopter.Image, imageX, imageY,
    imageW, imageH);
    if (imageY + imageH > pnlDisplay.Height)
    {
        imageY = pnlDisplay.Height - imageH;
        imageDir = -1;
    }
    else if (imageY < 0)
    {
        imageY = 0;
        imageDir = 1;
    }
}
```

In the top crossing code (the else if portion), we reset imageY to 0 (the top of the panel) and change imageDir to 1. Run the project again. Your image will now bounce up and down, beeping with each bounce, until you stop it. Stop and save the project.

The code we've developed here for checking and resetting image positions is a common task in Visual C# Express. As you develop your programming skills, you should make sure you are comfortable with what all these properties and dimensions mean and how they interact. As an example, do you see how we could compute imageX so the image is centered in the panel? Try this in the **tmrHover_Tick** method:

```
imageX = (int)(0.5 * (pnlDisplay.Width - imageW));
```

Put this line before the myGraphics.Clear line. Note the use of the cast (conversion) of the computation to an **int** type. Save the project one more time.

You've now seen how to do lots of things with animations. You can make images move, make them disappear and reappear, and make them bounce.

Image Erasure

In the little example we just did, we had to clear the panel control (using the **Clear** graphics method) prior to each DrawImage method. This was done to erase the image at its previous location before drawing a new image. This "erase, then redraw" process is the secret behind animation. But, what if we are animating many images? The Clear method would clear all images from the panel and require repositioning every image, even ones that haven't moved. This would be a slow, tedious and unnecessary process. It would also result in an animation with lots of flicker.

We will take a more precise approach to erasure. Instead of erasing the entire panel before moving an image, we will only erase the rectangular region previously occupied by the image. To do this, we will use the **FillRectangle** graphics method, a new concept. If applied to a graphics object named **myGraphics**, the form is:

```
myGraphics.FillRectangle(myBrush, x, y, width, height);
```

This line of code will “paint” a rectangular region located at (**x**, **y**), **width** wide, and **height** high with a brush object (**myBrush**).

And, yes, there’s another new concept – a **brush** object. A brush is like a “wide” pen. It is used to fill areas with a color. A brush object is declared (assume an object named **myBrush**) using:

```
Brush myBrush;
```

Then, a solid brush (one that paints with a single color) is created using:

```
myBrush = new SolidBrush(Color);
```

where you select the **Color** of the brush. Once done with the brush, dispose of the object using the **Dispose** method.

So, how does this work with the problem at hand? We will create a “blank” brush (we’ll even name it **blankBrush**) with the same color as the **BackColor** property of the panel (**myPanel**) control. The code to do this (after declaring the brush object) is:

```
blankBrush = new SolidBrush(pnlDisplay.BackColor);
```

Then, to erase an image located in **myGraphics** at (**imageX**, **imageY**), **imageW** pixels wide and **imageH** pixels high, we use:

```
myGraphics.FillRectangle(blankBrush, imageX, imageY, imageW, imageH);
```

This will just paint the specified rectangular region with the panel background color, effectively erasing the image that was there.

Add this line of code in the **general declarations** area:

```
Brush blankBrush;
```

Add this line in the **Form1_Load** method:

```
blankBrush = new SolidBrush(pnlDisplay.BackColor);
```

And, add this line in the **Form1_FormClosing** method:

```
blankBrush.Dispose();
```

These three lines declare, create and dispose of the brush object at the proper times. Finally, in the **tmrHover_Tick** method, replace the line using the Clear method with this new line of code (selective erasing):

```
myGraphics.FillRectangle(blankBrush, imageX, imageY, imageW, imageH);
```

Rerun the project. You probably won't notice much difference since we only have one object moving. But, in more detailed animations, this image erasing approach is superior.