Handling Events for Dynamically Created Controls – Click me if you can!

Last lesson we learned how to dynamically create, manage and destroy GUI controls. Are you ready for the next step? Good!

Wouldn't it be nice to be able to handle events for a dynamically created control? Wait, isn't that easy? You just go to the event tab in the Properties page … oh, yeah … you can only do that for a control that you are placing on the GUI with the Designer. Those types of controls are not dynamically created.

This is actually easier than you would imagine…just takes a few tricks. Let's get going…

To pull this off, what do we need to do? Two things:

1. Figure out how to create our own event handler
2. Figure out how to assign that event handler to the proper event for the control

The easiest way to do stuff like this is to see how the Visual C# IDE does it. Whenever you drag a control from the *ToolBox* onto the GUI in the *Design* view and change its properties, the IDE writes code for you to create and configure that control. The code it writes for you can be found in *Form1.Designer.cs*.

Let's drag a *PictureBox* onto our GUI and create a mouse click event handler for it. We will then go look in *Form1.Designer.cs* to see the code used to set up the event handler. Name the *PictureBox* ***myPictureBox***. When you create the mouse click event handler, the code window should open up with your cursor inside the newly created ***myPictureBox_MouseClick()*** event handler:

```
private void myPictureBox_MouseClick(object sender, MouseEventArgs e)
{
    |
}
```

We'll be coming back to this; for now, just remember the handler method's name: ***myPictureBox_MouseClick.***

Now find the code in *InititializeComponent()* that is setting this method as the mouse click event handler for the ***myPictureBox*** control (we've covered this earlier in the SPRITES – How to Make Them Move On Their Own lesson…if you need a refresher, go back to that lesson pages 2-6 and review). You should see the following:

```
this.myPictureBox.TabIndex = 3;
this.myPictureBox.TabStop = false;
this.myPictureBox.MouseClick += new System.Windows.Forms.MouseEventHandler(this.myPictureBox_MouseClick);
```

name of the control

name of the event

Method that is used to "connect" the event handler with the event

name of the event handler method

This line of code is connecting the event handler method (named ***myPictureBox_MouseClick***) to the mouse click event of the control named ***myPictureBox***. Don't go any further until you can make sense of the code above. We will basically replicate this line of code.

Handling Events for Dynamically Created Controls – Click me if you can!

Now let's create our own mouse click event handler. All we need to do is replicate the event handler the IDE created for us earlier. Here is the code again:

```
private void myPictureBox_MouseClick(object sender, MouseEventArgs e)
{
    |
}
```

Just copy this code and change the name of the method; let's rename it as ***my_MouseClick***:

```
private void my_MouseClick(object sender, MouseEventArgs e)
{

}
```

Next, we need to assign this event handler method as the one we want to use for our control. Do this in the constructor for the GUI as follows:

```
public Form1()
{
    InitializeComponent();

    myPictureBox.MouseClick += new MouseEventHandler(my_MouseClick);
}
```

Note that we removed all the *"this."* stuff and the "*System.Windows.Forms.*" stuff. These are redundant and unnecessary.

So, we now know how to assign our own event handler. Let's take the next step and figure out how to make this work with dynamically created controls.

Handling Events for Dynamically Created Controls – Click me if you can!

The first thing to note is we can share one event handler across many controls. Stated another way, multiple controls can share one event handler.

There is one challenge with this … if the event handler needs to do something with the control that generated the event, it's going to have to figure out which one of all the controls that use it is the one.

Let's look at our event handler again:

```
private void my_MouseClick(object sender, MouseEventArgs e)
{

}
```

Notice there are two arguments passed into the handler; these are the things in the parenthesis. The first is called ***sender*** and it holds the answer to our problem. It represents the control that caused the event and provides a method called ***Equal*()** that will tell us if the sending control matches a given, known control.

Suppose we have a *PictureBox* named ***pic1***. We can determine if the sender is ***pic1*** by:

```
private void my_MouseClick(object sender, MouseEventArgs e)
{
    if (sender.Equals(pic1) == true)
    {
        // pic1 is the control that caused the event!
    }
}
```

Once we figure out which control caused the event, we can do what we want with it, delete it for instance.

In our last lesson we learned how to use an array to hold all our dynamically created controls (*PictureBoxes* in that case). If we want each of these to respond to mouse clicks we will need to assign the mouse click event handler for each. Then in the mouse click event handler we will need to search through the array to see if the control that caused event matches one in the list. We'll use a ***foreach*** loop to accomplish this. Here is code using the setup from last lesson:

```
// this handles clicking in any PictureBox controlled by picList
private void my_MouseClick(object sender, MouseEventArgs e)
{
    foreach (PictureBox pic in picList)
    {
        if (sender.Equals(pic))
        {
            Controls.Remove(pic);    // GUI is no longer responsible for this control
            pic.Dispose();           // get rid of the actual control
            numPics--;               // update the count of controls in the list...one less
            DisplayBallCount();
        }
    }
}
```

So there you have it. Yell if you have any questions!