C# How-to 3: Adding event handlers to WinForms controls (part 1)

In How-to #2 we looked at creating a very simple WinForms application, including setting event handlers for the click event in two button controls. This is done easily by double-clicking the button, which automatically inserts a new event handler into the code and binds it to the button.

But what if you want to add handlers for events other than the click event? Let's say you want to add a mouseover event to button1. If you click the button in the design window, then go to the events panel in the properties window, you see the list of possible events. Click the MouseOver event: note that it has no event handler at present, and the righthand column is empty. If you click the arrow in the righthand column, a drop down list appears containing the names of two functions - button1_click and button2_click.

Now, what's going on here? These are handlers for the click events. How can you assign them to the mouseover event? Well, event handlers in .NET are known as delegates in C#. They are analogous in some ways to function pointers, or if you're a C++ programmer, callbacks. Essentially, such a function is delegated to be called when an event occurs without your having to call the function manually. In the .NET framework, each event has a delegate signature - that is, the handler function must have exactly the same parameters as those specified in the signature. The click event's signature has two parameters - an object (the sender object which sent the event) and a System.EventArgs object. The point is that any function with the same signature could be used as the handler for this event, even if it was not really intended for that purpose!

You'll have realised by now that the mouseover event delegate has the same signature as the click event, and therefore any function - which includes the click event handlers - could be used, and so the click event handlers are made available in the drop down list.

Now, it probably wouldn't be very useful to assign the click event handler to the mouseover event, but in other circumstances it could be. For example, at the moment we have separate event handlers for each button, but we could assign the same event handler to the click event for both buttons. Of course, then we would need to know which button generated the event, and this is where the sender object helps. Instead of two event handler functions, we could do this with just one:

```
private void button_Click(object sender, System.EventArgs e)
{
if (sender == button1)
MessageBox.Show("Hello World, from Microbion!!!", "First message", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
if (sender == button2)
this.close();
}
```

Using the property window, we could then assign the same event handler to both button1 and button2, and the handler would take action depending on the object which generated the event. This technique is known as multicasting.

Incidentally, a word of caution: say you set up an event handler for button1 and call the handler button1_Click. You might well want to rename the button at a later stage (to butStart, say). Of course, then butStart's event handler is called button1_Click, which isn't terribly helpful, so you rename the function in code to butStart_Click. If you compile the result, the compiler will flag an error, saying that function button1_Click is missing. In other words, Visual Studio does not detect the name change and automatically re-assign the correct handler. The workaround is to go back to the property window and select the correct handler from the drop down list in the event panel. You'll see that button1_Click is no longer in the list, but butStart_Click is.

In the next how-to, I will look at writing a mouseover and mouseleave events for the buttons, and look at assigning a keypress handler to the form.

C# How-to 4: Adding event handlers to WinForms controls (part 2)

In How-to #3 we saw how event handlers can be added to WinForms controls, and how one handler can be assigned to more than one control or event. We have also seen that you can assign a click event to a button by simply double-clicking the button. How to we assign another event handler - say for the mouseover event - to the control?

Go back to the project and click button1, then choose the events panel in the Properties window. If you look at the mouseover event, about halfway down in the Mouse section (it's actually called MouseHover - why couldn't Microsoft have stuck with the traditional JavaScript event names?) you'll see that it doesn't have a handler, yet. To add a handler is very simple - just double-click the event name in the left column of the window. If you do this, the code window opens and the cursor is in a new function:

```
private void button1_MouseHover(object sender, System.EventArgs e)
{

}
```

This has the same signature as the click event, as discussed in the previous how-to. Now we need to add something useful to prove it works - we'll change the button background colour when the mouse is over the control. This is easy. In the body of the handler, we just need the following code:

```
button1.BackColor=System.Drawing.Color.DeepPink;
```

which sets the control's background colour to deep pink (sorry if this clashes violently with your system!). This has a problem, though; when the mouse moves away from the control, the colour doesn't change back. So we need to add a MouseLeave event handler in the same way, and reset the colour. You should end up with:

```
private void button1_MouseLeave(object sender, System.EventArgs e)
{
button1.BackColor=System.Drawing.SystemColors.Control;
}
```

Note the use of System.Drawing.SystemColors rather than System.Drawing.Color to reset the control's background - this will reset it to whatever it was before it went pink.

Finally, let's add a handler to the form itself. If you click on the form and look at its events panel, find the KeyPress event in the Key section. If you click on the arrow in the right hand column, you see that there are no available event handlers. What happened to the two click and two mouse handlers?

The answer is that the KeyPress handler uses a different signature from the Click or MouseHover or MouseLeave handlers and therefore these handlers can't be assigned to the KeyPress event. Double-click on the KeyPress event name and you get a handler like this:

```
private void Form1_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
}
```

The sender object is still there, but the second parameter is different. Say we want to know what key was pressed. You could add this to the handler:

```
MessageBox.Show("You pressed this key: " + e.KeyChar, "Key press detected", MessageBoxButtons.OK,
MessageBoxIcon.Information);
```

All this does is get the ASCII character of the key pressed (if there is a printable character - try pressing the Escape key to see what happens) and shows it in a message box. Incidentally, if you're a VB programmer, you will know that to make this work the KeyPreview property of the form has to be set to True - if it doesn't work, check this has been done.

That's about it for event handlers. In the next how-to, I will start to look at how all this is put together to build an application that actually does something useful.