# Events and Delegates & Gui Programming

January 23, 2008

# Event Driven Programming

- Imagine a game with many actors responding to their environment.

# Event Driven Programming

- Imagine a game with many actors responding to their environment.
- Polling: Every so often, each actor looks at state of environment and takes appropriate actions.

# Event Driven Programming

- Imagine a game with many actors responding to their environment.
- Polling: Every so often, each actor looks at state of environment and takes appropriate actions.
- Events: Wake up actors when something interesting happens.

# We can code events using basic C#...

```
public interface IEventHandler{ void DoIt(); }

public class SesameStreet{
  void RegisterForCookie(IEventHandler h){...}
  void CookieEventHappens(){...}
}

public class CookieMonster{
  class CookieEatingClass : IEventHandler{
      void DoIt() { WriteLine("Nom, Nom"); }
  }

  CookieMonster(SesameStreet s){
      s.RegisterForCookie(new CookieEatingClass());
  }
}
```

# . . . but the encoding is flawed.

Problems:

- A nested class is needed to define each event handler.
- Handler has not easy access instance and local variables.
- Resulting code is hard to read.

# Delegates: methods as data.

```
// Declare a new delegate type.  A binOp is a
// method that takes two ints and returns an int.
public delegate int binOp(int x, int y);

public class Demo{

    static void Main(string[] args){
        // m is stores a binOp
        binOp m = Math.Min;

        // Calling m calls the stored method,
        // Math.Min. Output is "3".
        Console.WriteLine(m(3,4));
    }
}
```

# Multicasting: A delegate can call several methods. (I)

```
public delegate void Printer(string s);

public class PromptPrinter{
    private string prompt;
    public PromptPrinter(string p){ prompt=p; }
    public void Print(string s){
        Console.WriteLine(prompt + s);}
}
```

# Multicasting: A delegate can call several methods. (II)

```
public class Demo{
    static Printer myPrinter;

    static void Main(string[] args){
        PromptPrinter p1 = new PromptPrinter(">>");
        PromptPrinter p2 = new PromptPrinter("#");
        myPrinter = p1.Print;
        myPrinter += p2.Print;
        myPrinter("foo");
    }
}
```

- Output is "»foo" "#foo"
- Multicasting only makes sense for methods returning void.
- Operators =, +, -, +=, -= attach and detach delegates.

# Anonymous delegates further streamline event code.

```
public delegate int binOp(int x, int y);
   ...
// C# 2.0 "Anonymous Delegate" Syntax:
binOp sum =
  delegate(int x, int y) {
    return x + y; };

// C# 3.0 "Lambda" Syntax
// (plus type inference):
binOp sum = ((x, y) => x + y );
```

# How does it work?

- C# compile translates delegate types into classes which inherit from System.MulticastDelegate.
- Delegate values are compiled to class instances.
- For multicasting, + operator builds a list of delegates objects.

# Events are delegates with standard method signatures.

```
public delegate HandlerType(object caller,
                            EventArgs e);

class foo{
  public event HandlerType myEvent
}
```

- Member eventName can be updated (+, +=, ...) as public.
- But, eventName can only be invoked by foo.
- By convention, foo should pass itself as caller.

# Updating the cookie example (I)

```
class CookieEventArgs : System.EventArgs { };

class SesameStreet{

  delegate void CookieDelegate(object o,
                               CookieEventArgs c);
  event CookieDelegate CookieEvent;
  void DoCookie() {
      CookieEvent(this, new CookieEventArgs());}
}
```

# Updating the cookie example (II)

```
class CookieMonster{

  CookieMonster(SesameStreet s){
  s.CookieEvent +=
    ((object o, CookieEventArgs c) =>
      System.Console.WriteLine("Nom, Nom") );
  }
}
```

# Updating the cookie example (III)

```
public class Runner{
    static void Main(string[] args)
    {
        SesameStreet ss = new SesameStreet();
        CookieMonster cm = new CookieMonster(ss);
        ss.DoCookie();
        ss.DoCookie();
        ss.DoCookie();
    }
}
/* Output:     Nom, Nom
               Nom, Nom
               Nom, Nom      */
```

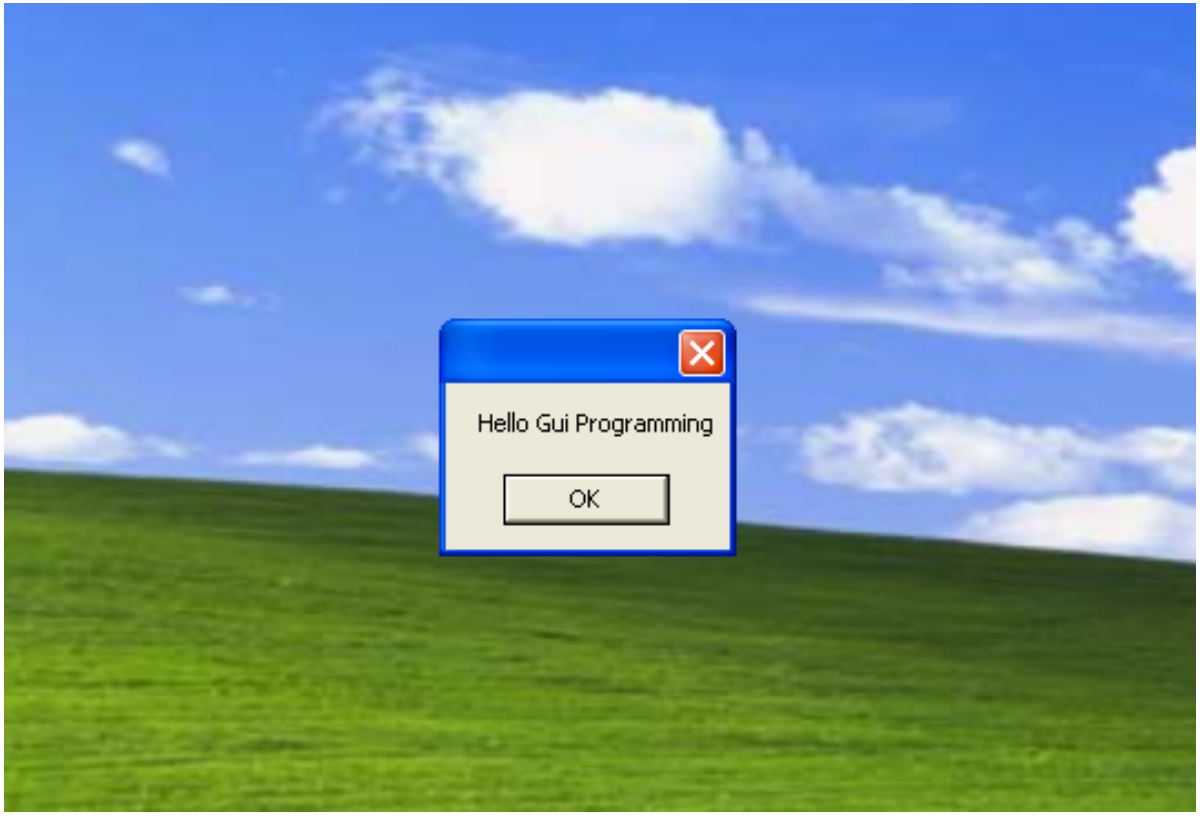
# Gui programs are not special.

- Execution starts at Main
- Events model used to get inputs from controls
- Fancy designers just a convenient way to generate code
- (One caveat coming up)

# A Simple Gui Program (I)

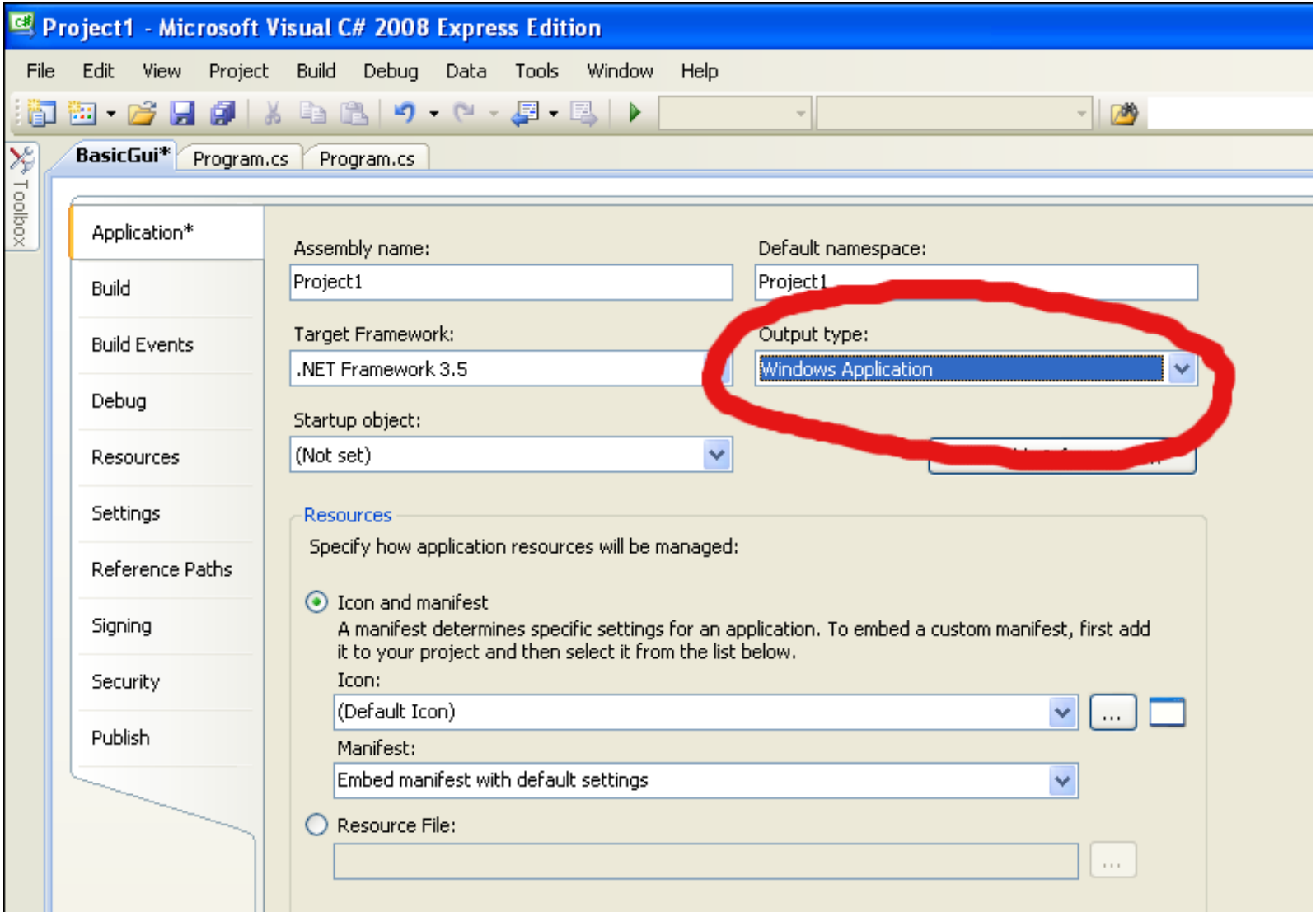
```
using System.Windows.Forms;

// Simplest GUI program.
// Compile as a "Windows Application"
class Program
{
    static void Main(string[] args)
    {
        MessageBox.Show("Hello Gui Programming");
    }
}
```

# A Simple Gui Program (II)



# A Simple Gui Program (III)

The caveat: I had to change the project's output type to "Windows Application". This stops the program from popping up a command prompt.



# Event Driven Gui Programming

- All screen elements are represent by objects.
- Interesting user activities trigger events.
- Handling these event lets your program update it's state.

- Windows are instances of System.Windows.Forms.Form
- Buttons are instances of System.Windows.Controls.Button

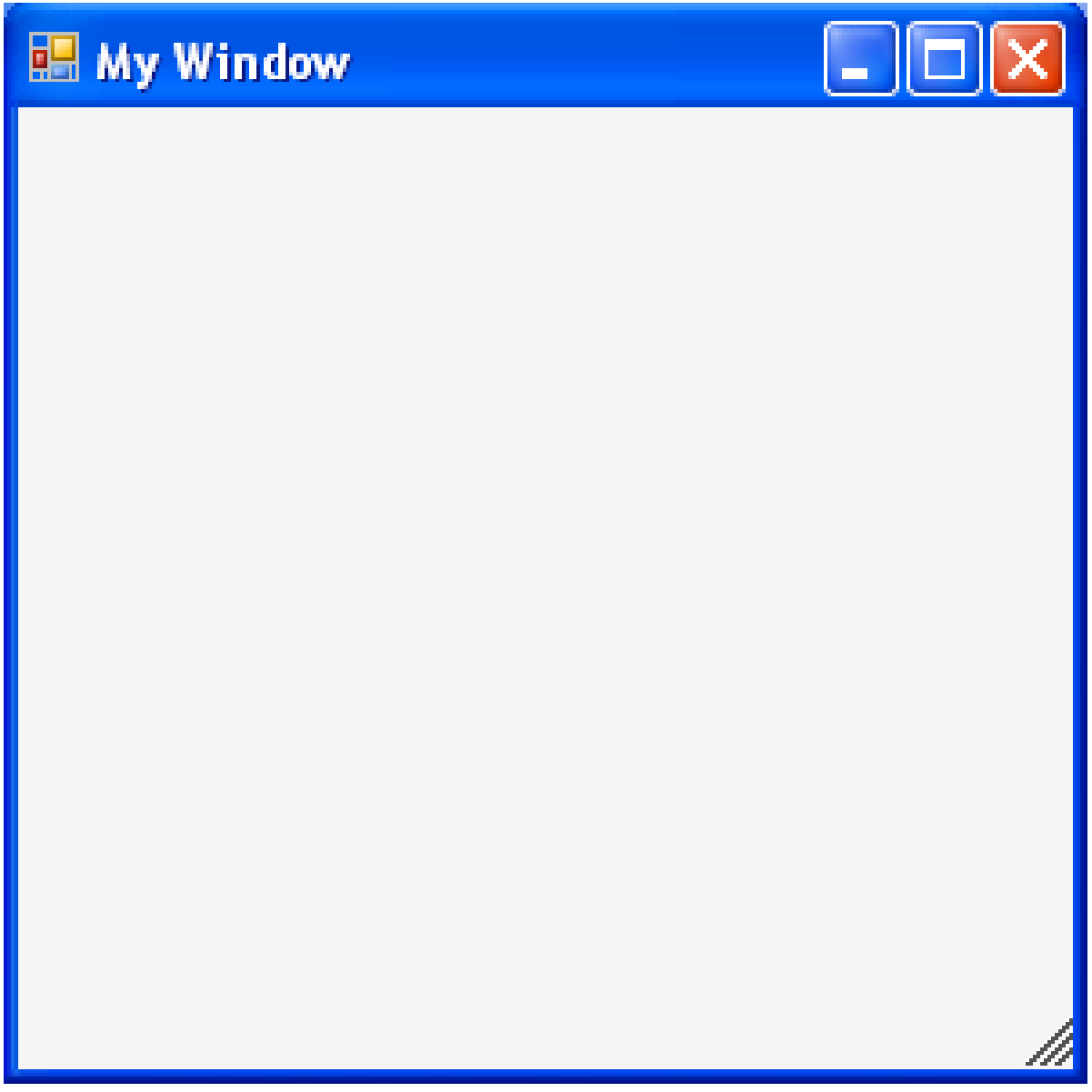
# Finally: A Gui That Does Something! (I)

```
static void Main() {
    RandColorPicker cp = new RandColorPicker();

    Form theForm = new Form();
    theForm.Text = "My Window";

    //Event handlers here
    theForm.MouseClick +=
      ((x,y) => theForm.BackColor = cp.GetRand());
    theForm.MouseEnter +=
      (delegate(object x, EventArgs y) {
        theForm.BackColor = cp.GetRand(); });

    theForm.ShowDialog();
}
```

# Finally: A Gui That Does Something! (II)

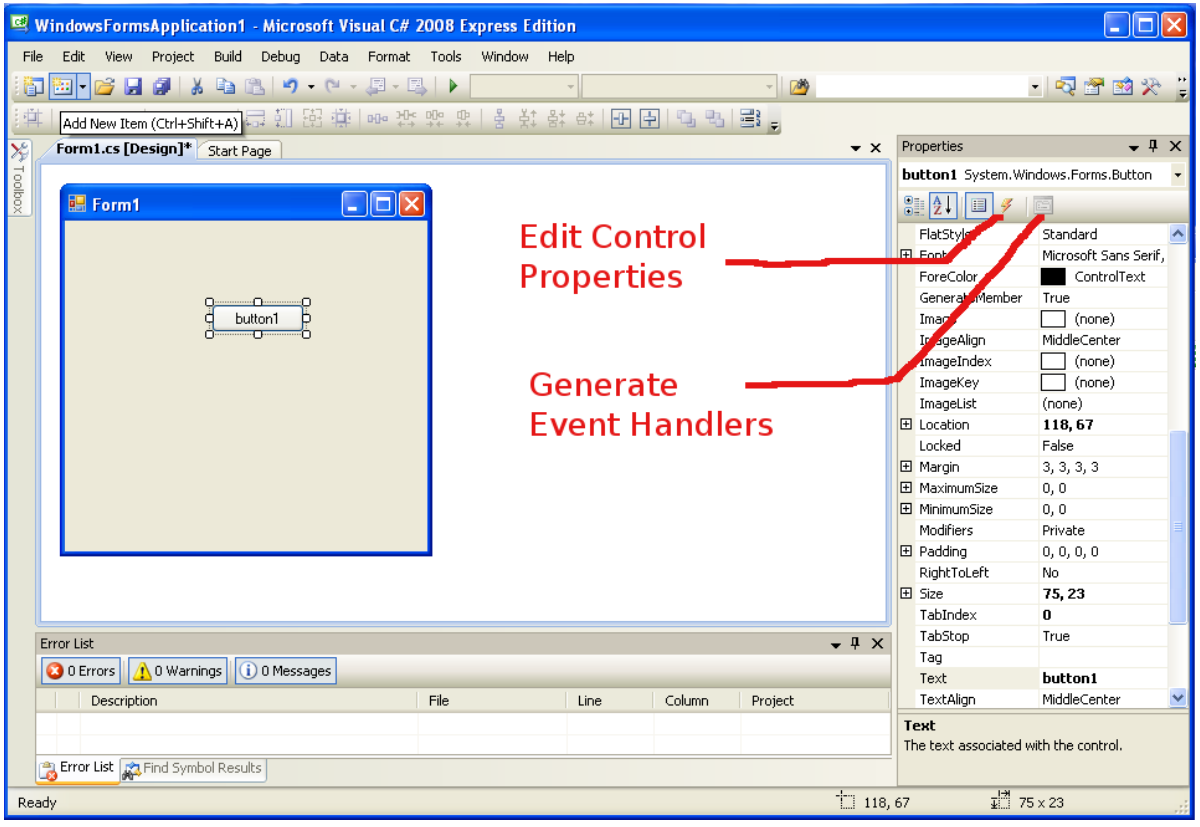
# Finally: A Gui That Does Something! (II)

# Finally: A Gui That Does Something! (II)



# Visual Studio's form designer helps build Guis.

- Drag and drop controls onto forms.
- Designer can set properties of controls.
- Designer can automatically generate stub code for event handlers.



# Homework

- Problem Set 1: Due before class today(!)
- Problem Set 2
  - Will be posted by the end of the day tomorrow
  - PS 2 will be with partners. Email me by Friday. If you have a partner, tell me who. If you don't, let me know and I'll pair people up. There may need to be one group of three.
  - Due February 6.
  - Start early.