A Tour of Windows Forms

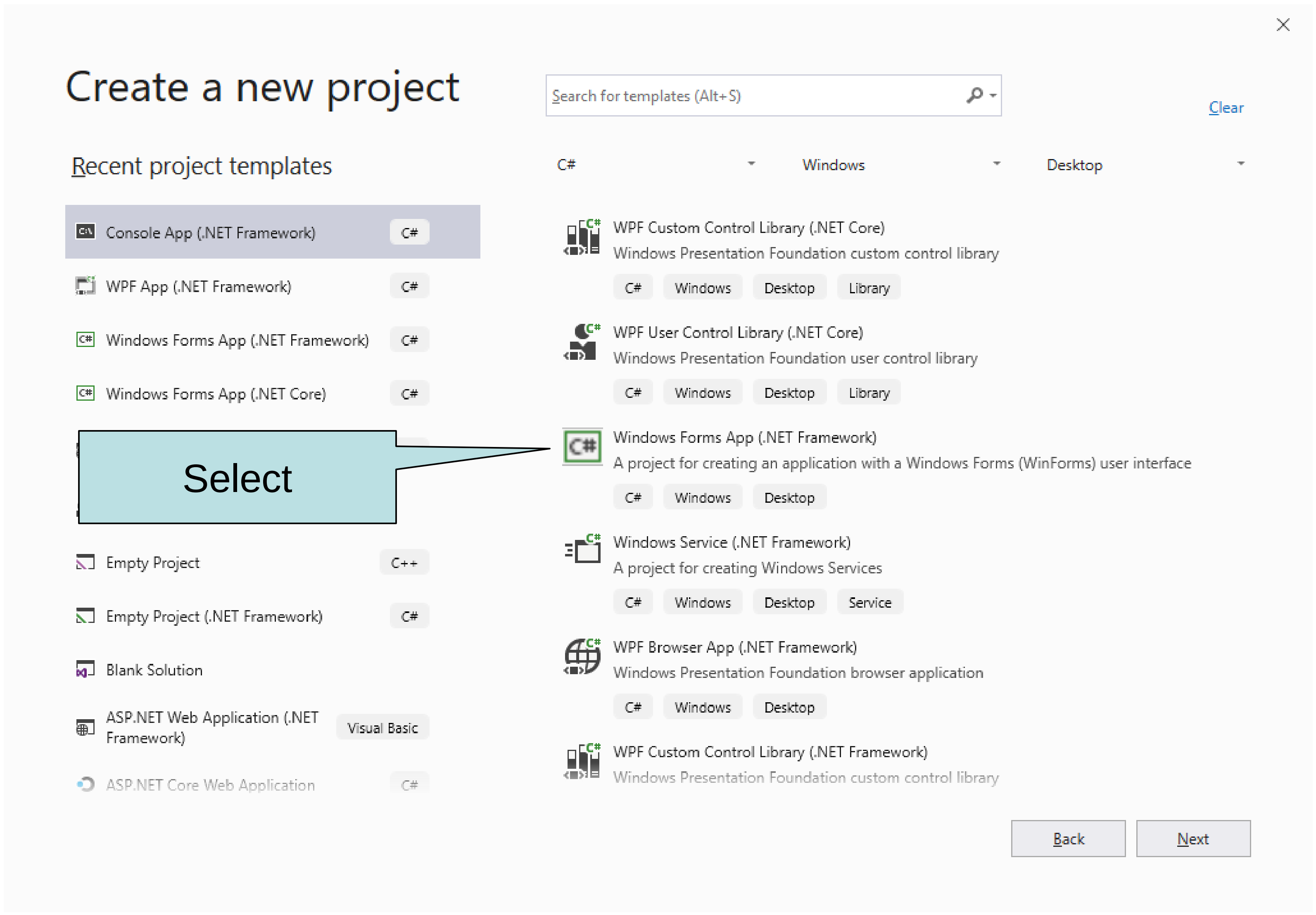
Development Frameworks

- Windows Forms
  - Original implementation
  - Uses code to render the GUI
  - Uses Windows GDI+ for graphics
- Windows Presentation Foundation (WPF)
  - Newer, but more difficult to learn
  - Uses markup (XAML) to render the GUI
  - Uses DirectX for graphics

Course Choice is Windows Forms

- Still very popular
- Skills more easily transferred to other platforms
- Very good support with VS designer for GUI layout
- Close coupling with C# code
- Quicker path to developing applications geared to learning

Create New Project



Create a new project

Search for templates (Alt+S)

Clear

Recent project templates

Console App (.NET Framework)	C#
WPF App (.NET Framework)	C#
Windows Forms App (.NET Framework)	C#
Windows Forms App (.NET Core)	C#
Empty Project	C++
Empty Project (.NET Framework)	C#
Blank Solution	
ASP.NET Web Application (.NET Framework)	Visual Basic
ASP.NET Core Web Application	C#

C# | Windows | Desktop

WPF Custom Control Library (.NET Core)
Windows Presentation Foundation custom control library
C# Windows Desktop Library

WPF User Control Library (.NET Core)
Windows Presentation Foundation user control library
C# Windows Desktop Library

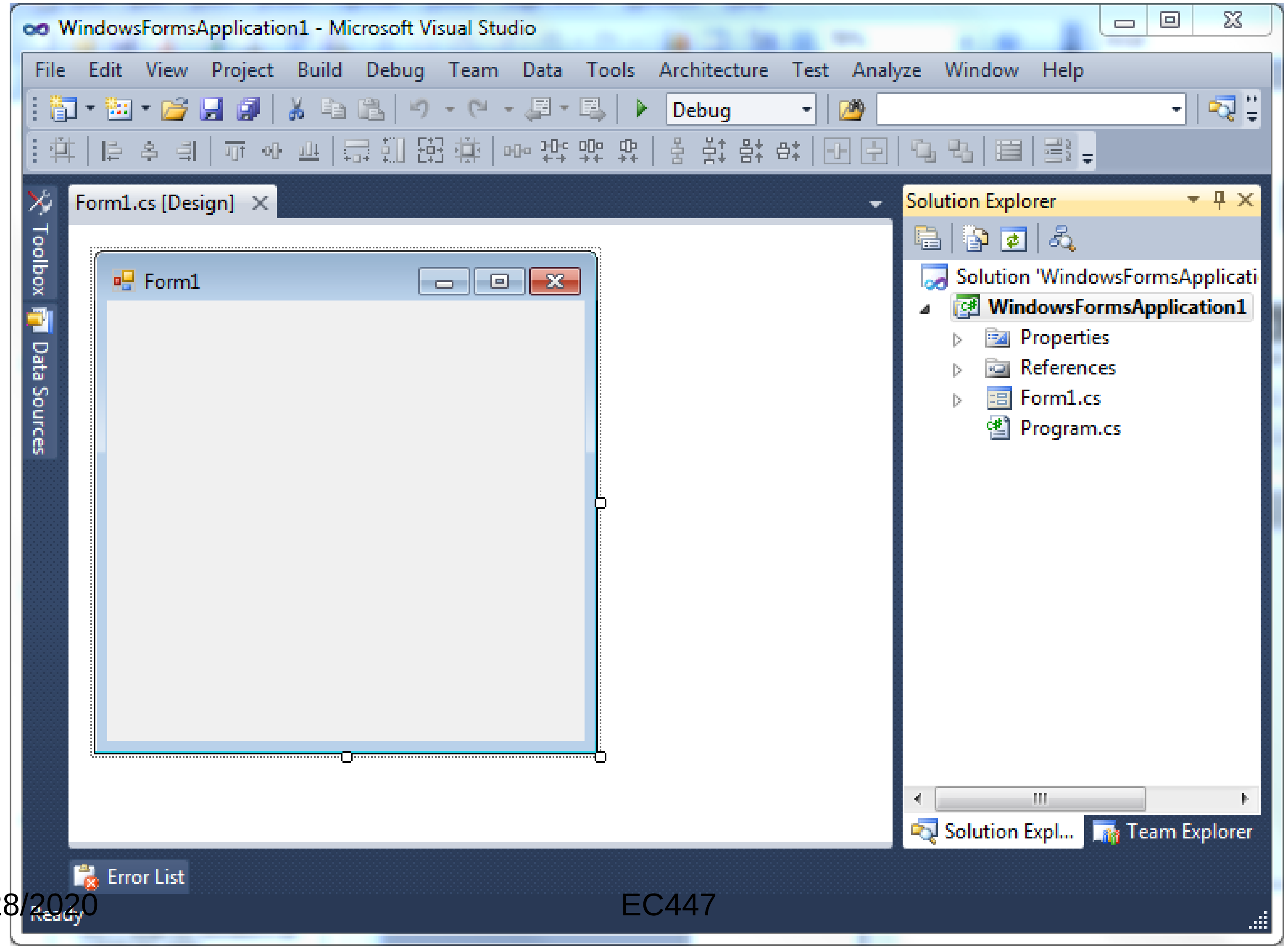
Windows Forms App (.NET Framework)
A project for creating an application with a Windows Forms (WinForms) user interface
C# Windows Desktop

Windows Service (.NET Framework)
A project for creating Windows Services
C# Windows Desktop Service

WPF Browser App (.NET Framework)
Windows Presentation Foundation browser application
C# Windows Desktop

WPF Custom Control Library (.NET Framework)
Windows Presentation Foundation custom control library

New Windows Forms Application


WindowsFormsApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Toolbox
Data Sources
Form1.cs [Design]
Form1
Solution Explorer
Solution 'WindowsFormsApplicati
WindowsFormsApplication1
Properties
References
Form1.cs
Program.cs
Solution Expl...
Team Explorer
Error List
Ready

Generated Code

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Partial Classes

- Allow breaking a class into multiple files.
- Allow generated code to be hidden from the user provided code.
- Note the default namespaces provided.
- *Form1* is derived from *Form* which in turn is derived from *Control.*
- A Control has an associated *window* in the operating system.

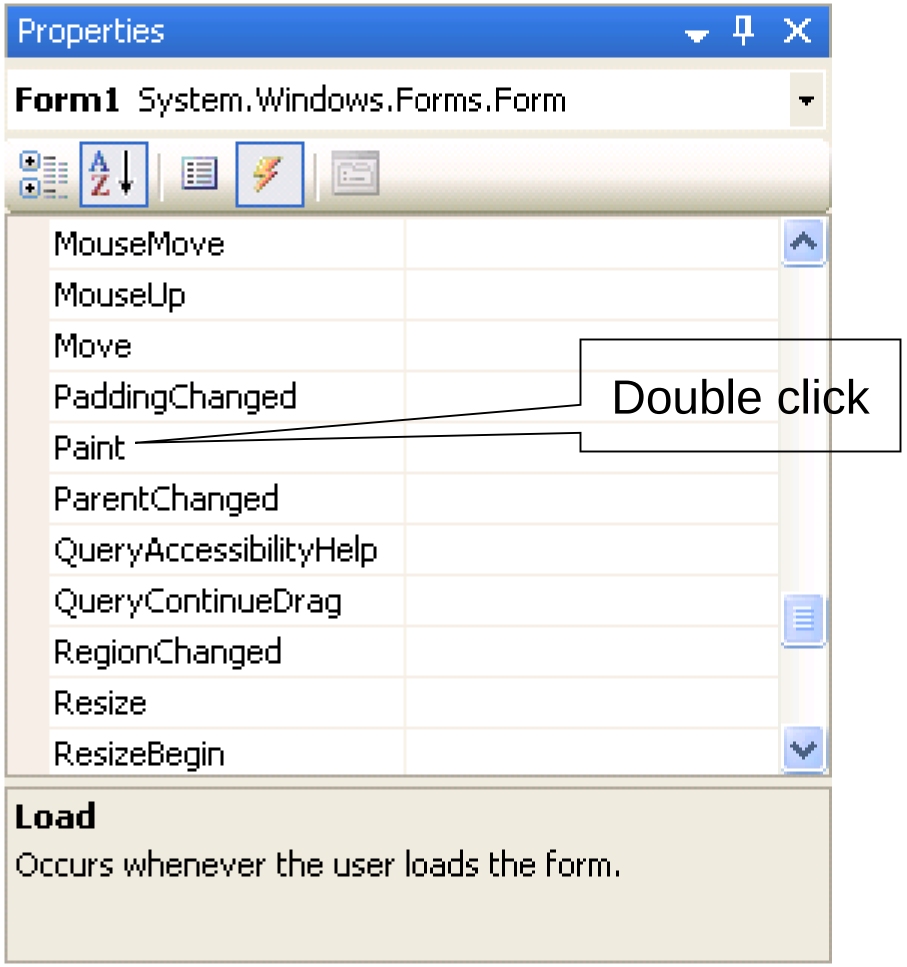
Paint Events

- Windows messages are passed to your program by events.
- Predefined *delegates* are defined for almost all the primary windows messages.
- Accessed by adding an event handler or by overriding the appropriate event handler in the base class.

When Paint Events are Generated

- Restoring a window from the task bar.
- Uncovering a portion of a window that is covered by another window.
- At the request of your program itself.
- Moving a form does **not** generate an event unless it also brings a form to the top and thus uncovering part of it.

Adding a Paint Event Handler


Properties

Form1 System.Windows.Forms.Form

MouseMove
MouseUp
Move
PaddingChanged
Paint
ParentChanged
QueryAccessibilityHelp
QueryContinueDrag
RegionChanged
Resize
ResizeBegin

Load
Occurs whenever the user loads the form.

Empty Event Handler Method

```
private void Form1_Paint(object sender, PaintEventArgs e)
   {
   }
```

This arg is important

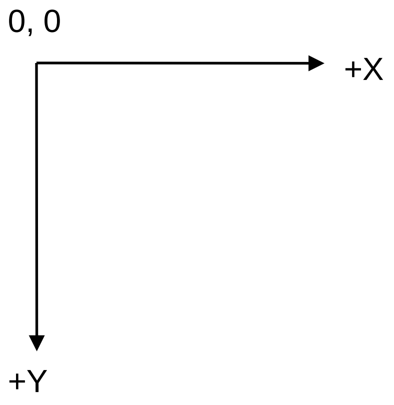
The Graphics Object

- In order to “paint” in the form we need a graphics object.
- The “client area” is the area inside the borders of the form that we can output to.
- The graphic object obtains a “device context” from the Win32 operating system API.
- We use the methods in the Graphics class to perform output.
- The are a myriad of methods available in the Graphics class.

Getting the Graphics Class Reference

```
  private void Form1_Paint(object sender, PaintEventArgs e)
         {
             Graphics g = e.Graphics;
         }
    }
```

Default Coordinate System

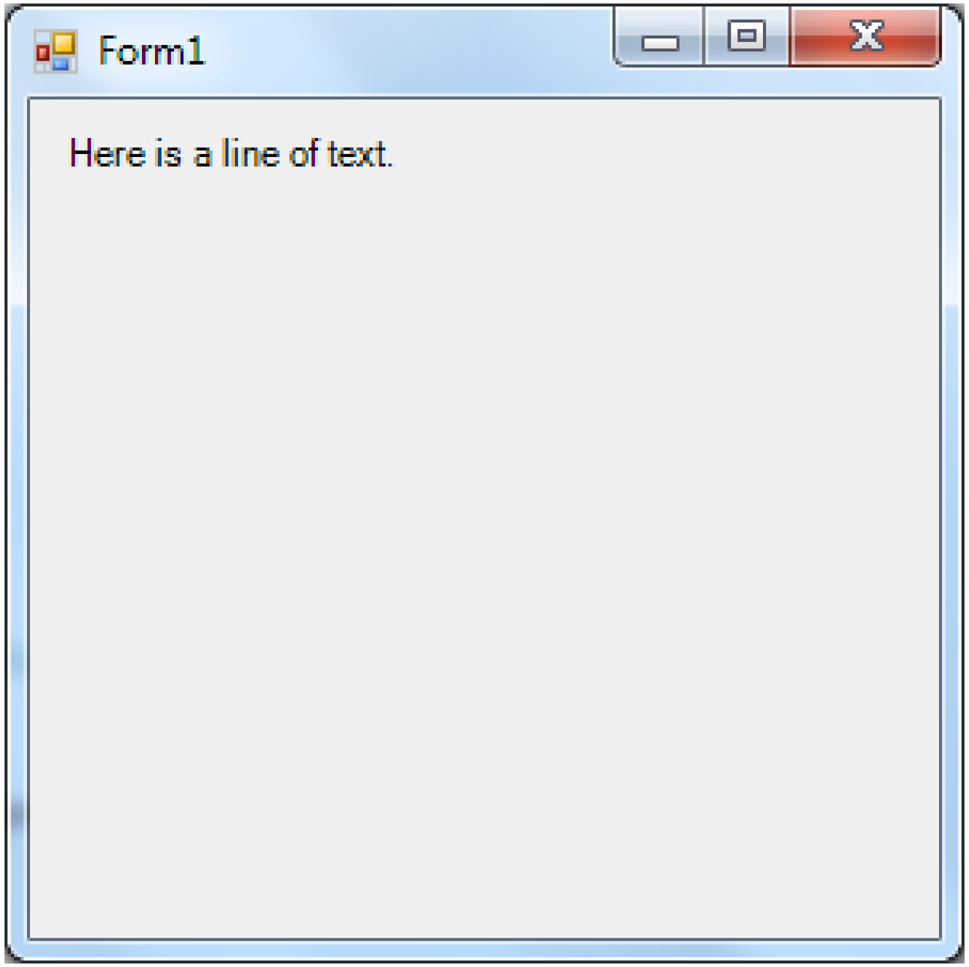


The drawing unit is the pixel

Displaying Text

```
    private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.DrawString("Here is a line of text.", Font,
               Brushes.Black, 10, 10);
        }
```

The Output



```
Here is a line of text.
```

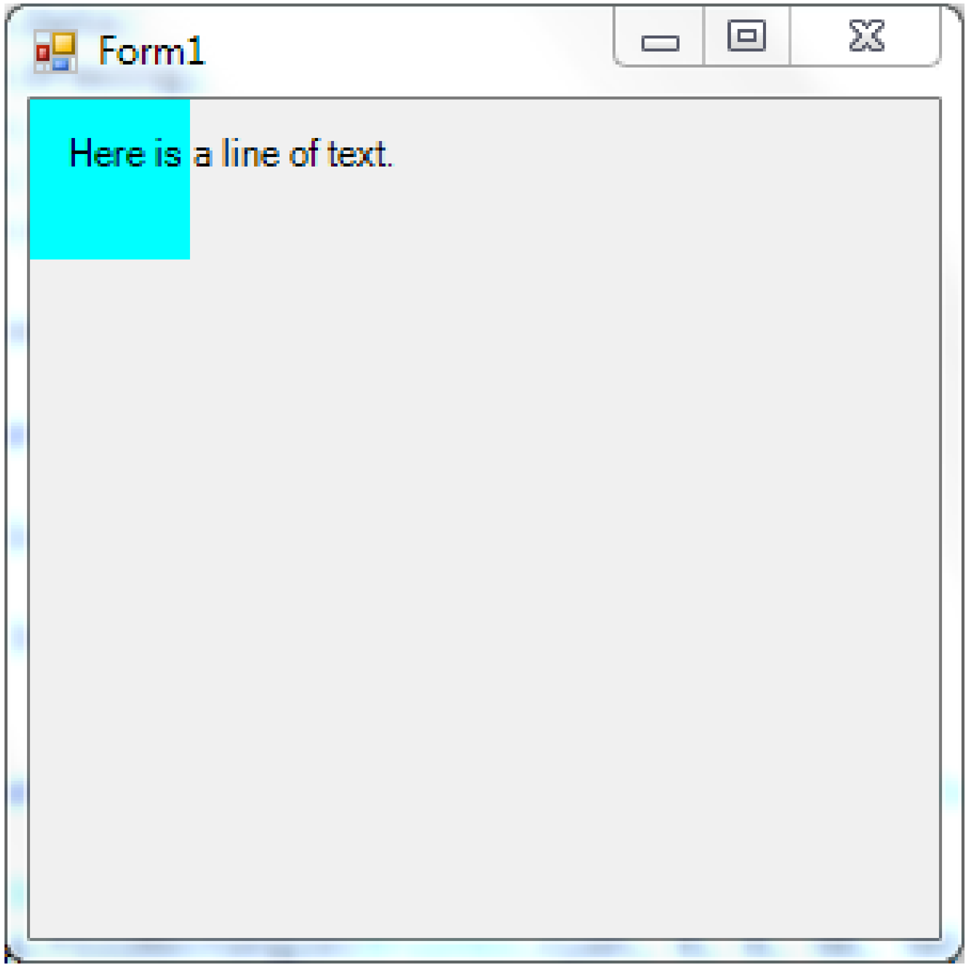
Brushes and Fonts

- A *Brush* class object is used to draw the text (or graphics as we will see).
- There are many built in brushes in the static *Brushes* class.
- A Font is used to determine the look of the characters, e.g., the *typeface*.
- In the example I use the default *Font* object from the form itself.
- This can be confusing as the same identifier can be used for a *class* and a *property*. The context determines which one it is.

More Complicated Program

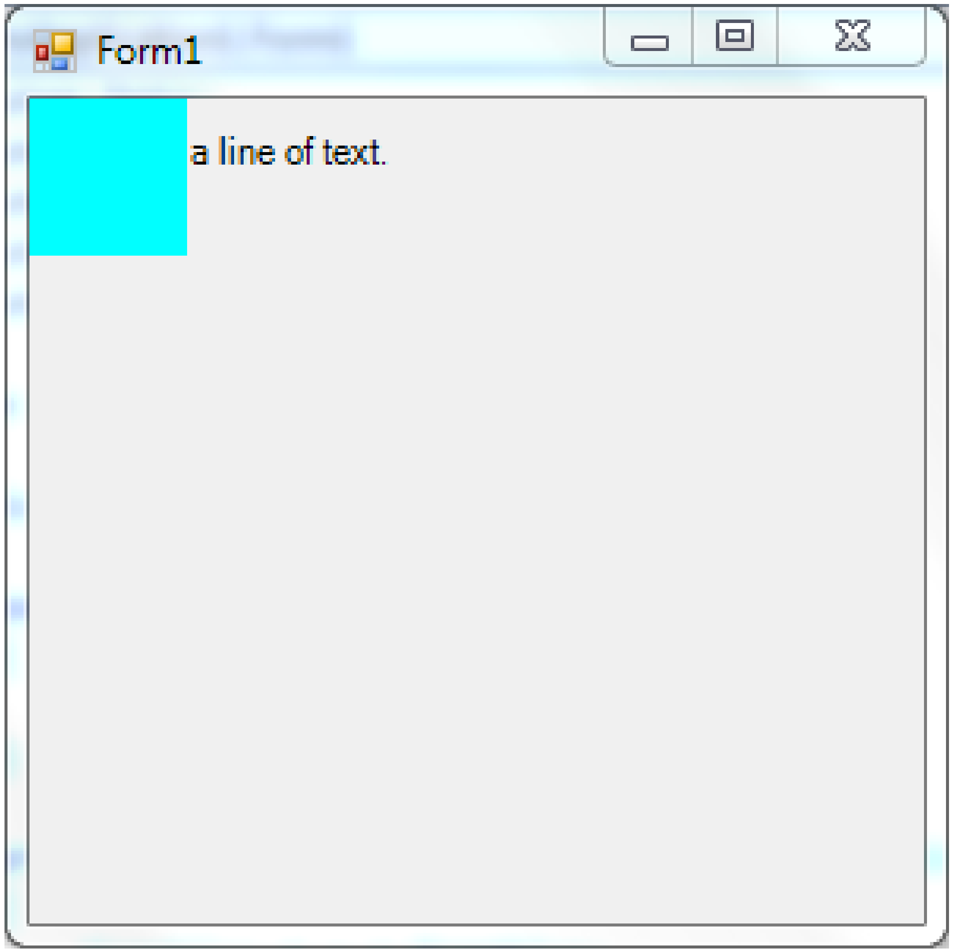
```
private void Form1_Paint(object sender, PaintEventArgs e)
     {
          Graphics g = e.Graphics;
          g.FillRectangle(Brushes.Cyan, 0, 0, 50, 50);
          g.DrawString("Here is a line of text.", Font,
              Brushes.Black, 10, 10);
     }
```

The Output



Here is a line of text.

Reversing the Z-order



a line of text.

The Mouse

- I will cover the mouse in much greater detail in another chapter.
- We can determine which button was pressed and the coordinate of the click in display units.
- We need to add an appropriate event handler to process the event.
- Use the *MouseClick* event for now. There are several other mouse events we will use later/

A Mouse Event Handler

```
private void Form1_MouseClick(object sender,
    MouseEventArgs e)
    {
            int x, y;
            x = e.X;
            y = e.Y;
            if (e.Button == MouseButtons.Left)
            {
                //do something for left button
            }
    }
```

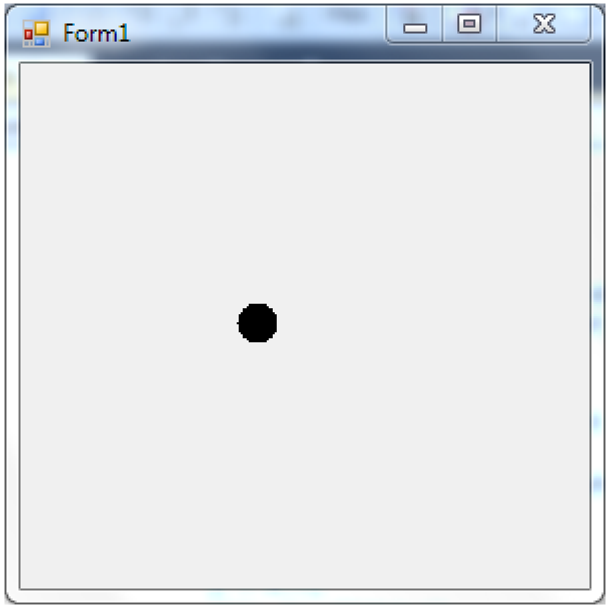
Mouse Click Example

- We draw a small filled circle (ellipse) wherever the mouse is clicked.
- Use *Invalidate* to cause a paint event.
- *X* and *y* must be *fields* and not local variables.
- Why?

Mouse Click Example

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace MouseClick
{
    public partial class Form1 : Form
    {
        private int x;   //mouse x coord
        private int y;   //mouse y coord
        public Form1()
        {
            InitializeComponent();
        }
private void Form1_MouseClick(object sender,
    MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                x = e.X;
                y = e.Y;
                this.Invalidate();
            }
        }
        private void Form1_Paint(object sender,
              PaintEventArgs e)
        {
            const int WIDTH = 20;
            const int HEIGHT = 20;
            Graphics g = e.Graphics;
            g.FillEllipse(Brushes.Black, x-WIDTH/2,
                y-WIDTH/2, WIDTH, HEIGHT);
        }
    }
}
```

The Output



Making the Program Better

- We can only display one circle, the last position clicked.
- What if we were to draw the circles within the mouse event handler? We could do that except the contents of the client area would be lost if the form needed repainting.
- Instead lets use a versatile data structure in the FCL, the *ArrayList*.
- An *ArrayList can dynamically* grow and store any class derived from *Object*. This is, of course, everything!

Better Program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
namespace MouseClick
{
    public partial class Form1 : Form
    {
        private ArrayList coordinates = new ArrayList();
        public Form1()
        {
            InitializeComponent();
        }
```

Add this namespace.

```
private void Form1_MouseClick(object sender,
  MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                Point p = new Point(e.X, e.Y);
                this.coordinates.Add(p);
                this.Invalidate();
            }
        }
        private void Form1_Paint(object sender,
          PaintEventArgs e)
        {
            const int WIDTH = 20;
            const int HEIGHT = 20;
            Graphics g = e.Graphics;
```

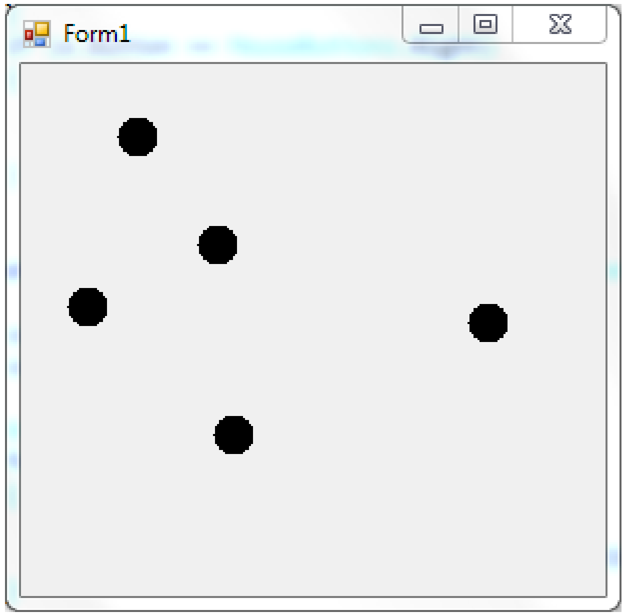
The use of the *this* keyword is not necessary except in the case of an ambiguous reference such as between a method parameter and a field.

```
 foreach (Point p in this.coordinates)
                {
                    g.FillEllipse(Brushes.Black,
                        p.X-WIDTH/2, p.Y-WIDTH/2,
                        WIDTH, HEIGHT);
                }
            }
        }
}
```

Foreach

- *Foreach* is a looping control structure not found in C or C++.
- It is much easier to use than a *for* loop, but only works with data structures that support certain interfaces such as *IEnumerable*.
- We will talk at length about this in a subsequent chapter. All common collection classes in the FCL work with *foreach* as well as all arrays.
- A simple array is actually an instance of the *Array* class. *Array* is simpler than the *ArrayList* class and is not growable.

The Output



Clear List with Right Button

```
private void Form1_MouseClick(object sender,
        MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        Point p = new Point(e.X, e.Y);
        this.coordinates.Add(p);
        this.Invalidate();
    }
    if (e.Button == MouseButtons.Right)
    {
        this.coordinates.Clear();
        this.Invalidate();
    }
}
```

Clears the entire Arraylist

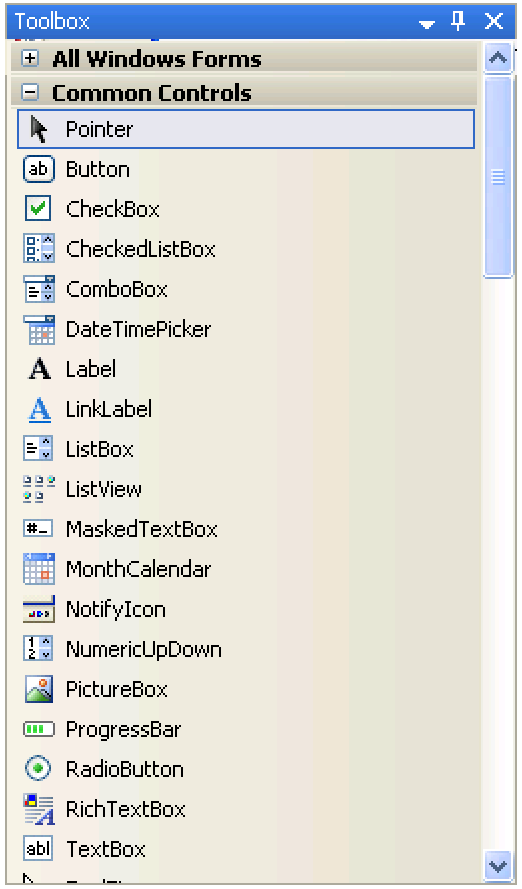
Controls

- Controls are special windows and have associated classes in .NET.
- Buttons
- Check boxes
- Edit controls
- Radio buttons
- List controls

The Toolbox

- The toolbox is used to *drag and drop* controls onto your form.
- The *designer* generates the code necessary to place these controls on your form at run time.
- Controls are painted automatically.
- You interface with your controls using the properties and methods of the respective class.

The Toolbox



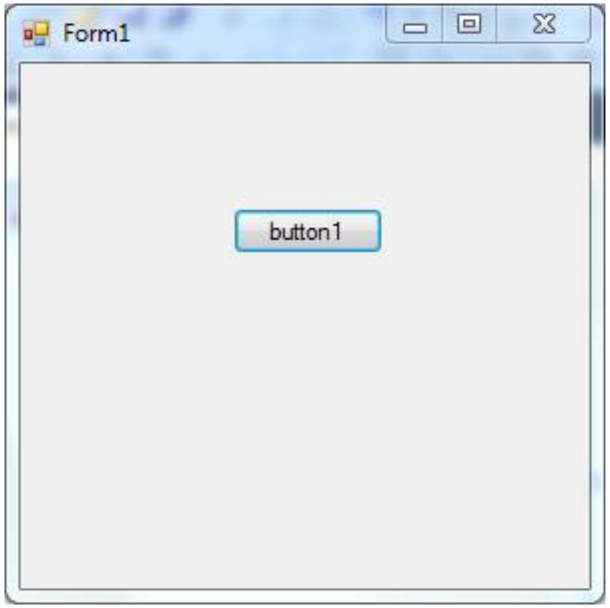
Toolbox

All Windows Forms

Common Controls

- Pointer
- Button
- CheckBox
- CheckedListBox
- ComboBox
- DateTimePicker
- Label
- LinkLabel
- ListBox
- ListView
- MaskedTextBox
- MonthCalendar
- NotifyIcon
- NumericUpDown
- PictureBox
- ProgressBar
- RadioButton
- RichTextBox
- TextBox

Adding a Button



Using the Button

- Open the properties window.
- Change the label by using the *Text* property.
- Add an event handler for the *Click* event.
- Shortcut – double click on the control.
- To clear the circles from our last example add the following code:

```
private void button1_Click(object sender, EventArgs e)
        {
            this.coordinates.Clear();
            this.Invalidate();
        }
```

The Designer’s Hidden Code

- The designer code is placed in a related file that uses the partial class feature.
- It has the same name with *designer* added.
- For example, *Form1.Designer.cs*.

Example of Designer Generated Code

```
namespace MouseClick
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Windows Form Designer generated code
```

Example – contd.

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(95, 30);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 0;
            this.button1.Text = "Clear";
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
```

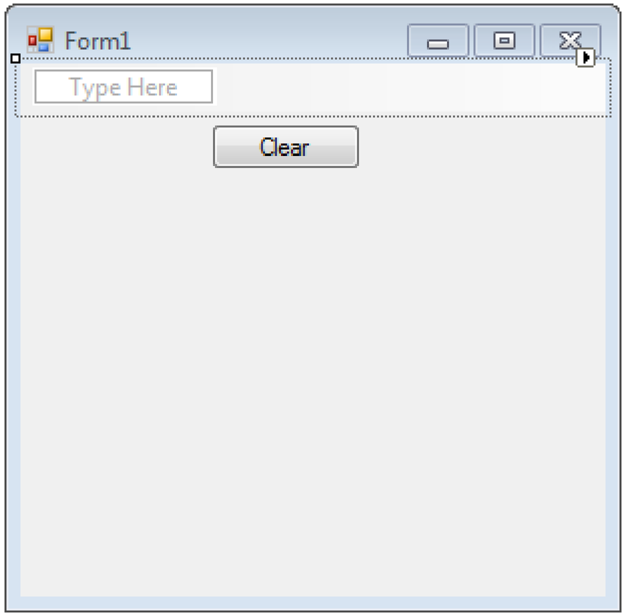
Example – contd.

```
 this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(292, 266);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.Paint += new System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
            this.MouseClick += new
System.Windows.Forms.MouseEventHandler(this.Form1_MouseClick);
            this.ResumeLayout(false);
        }
        #endregion
        private System.Windows.Forms.Button button1;
 }
```

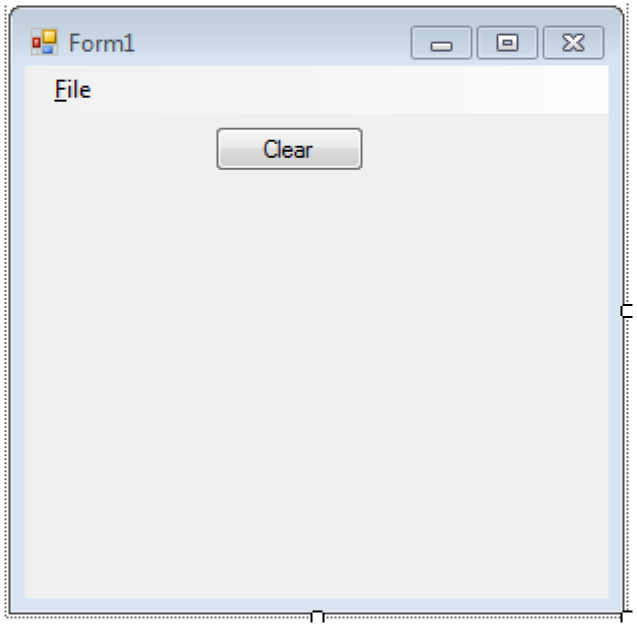
Adding a Menu

- The .NET Framework 2.0 has a new menu class, *MenuStrip*. It replaces the older *MainMenu* class.
- Drag a *MenuStrip* from the toolbox.
- Add items to the menu by just typing them into the placeholders. The designer generates the code.

Form with MenuStrip



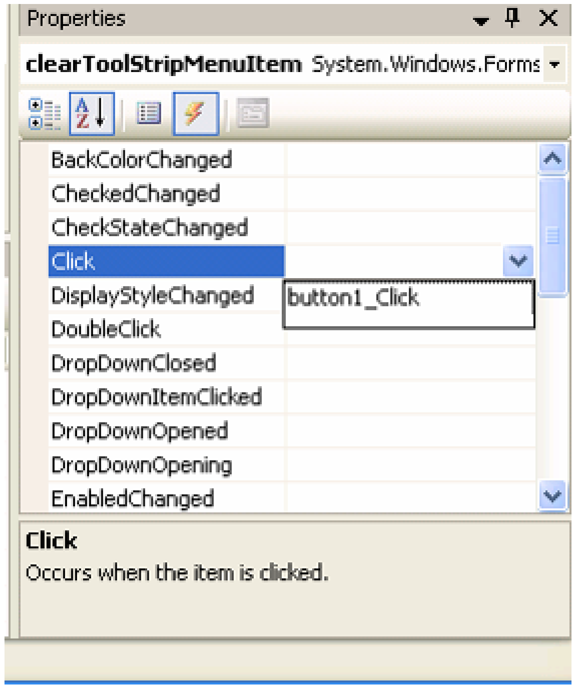
Adding a Menu Item



Adding an Event Handler

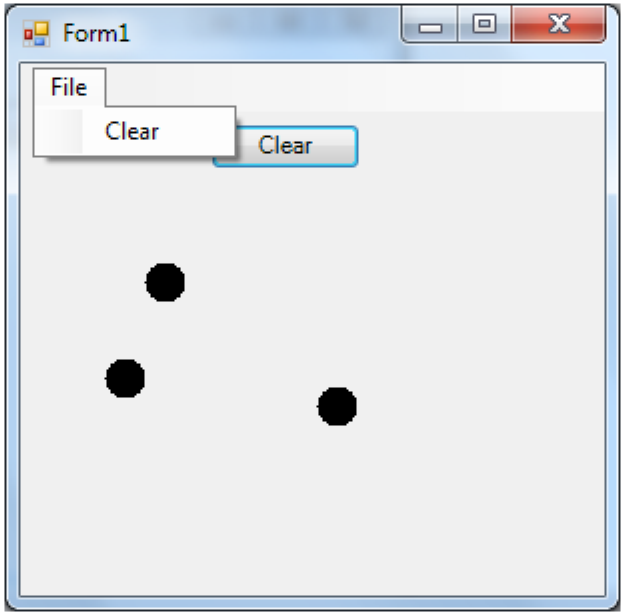
- Use the properties window after selecting the item on the menu.
- Add a handler for the *Click* event.
- Add code to the handler just as you did for the mouse or paint event.

The Resulting Application



BackColorChanged	
CheckedChanged	
CheckStateChanged	
Click	
DisplayStyleChanged	button1_Click
DoubleClick	
DropDownClosed	
DropDownItemClicked	
DropDownOpened	
DropDownOpening	
EnabledChanged	

Click
Occurs when the item is clicked.



We now have three ways to clear the form.

Summary

1. Creating a Windows Forms application using the wizard.
2. Adding a *Paint* event to allow text and graphics to be displayed in the client area of the form.
3. Drawing text and simple graphics.
4. Handling the *MouseClick* event.
5. The concept of controls.
6. Adding a *Button* control and its *Click* event handler to the form.
7. How the designer manages hidden code.
8. Adding a menu and processing *MenuItem* events.