University of Hull

Windows Presentation Foundation (WPF) User Interfaces

Rob Miles

Department of Computer Science

29c 08120 Programming 2

Design Style and programming

- As programmers we probably start of just worrying about making the program work
  - This is a very good place to start
- But in modern systems the "look and feel" of the user interface is very important
  - No matter how good the code is, if the program is hard to use it will not be popular
- You should pay careful attention to user interface issues when making your programs

Separating User Interface Design and Code

- It turns out that programmers are not always very good at graphic design
  - And that graphic designers are not very good at programming
- To make a good application we need a good user interface design and code that works
- It makes sense to separate programming and design and make it easy for the graphic designer and the programmer to work together

Windows Presentation Foundation (WPF)

- The Windows Presentation Foundation separates the user interface design from the program code by the use of a "markup" language called XAML
  - This stands for "eXtensible Application Markup Language"
  - It describes the arrangement of items on a window
- The designer can create the XAML and the programmer can use the objects defined in it to create the code
- Visual Studio provides an environment where the XAML and the program can be worked on together

XAML and Windows Presentation Foundation

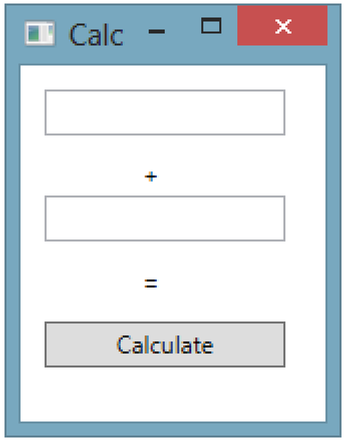
- A WPF application is made up of pages that contain elements
- These have properties that determine where they are, how they appear and what they can do in an application
- The Visual Studio tool allows us to manipulate the page content by using the design surface and the element properties pane

Expressing WPF Elements

- The description of the elements in a WPF application is actually held in a text file
- This file is formatted in a particular way
- Microsoft invented a language, XAML to hold this design information:
  - eXtensible Application Markup Language
- XAML was invented to hold user interface design information
- It is based on the XML standard

Why do we need XAML?

- XAML allows us to separate the role of graphic designer and programmer
  - The designer should not have to see code objects to work
  - The programmer should not be held back while the design is produced
- The XMAL file provides a separation between the code that drives the application and the way the application looks

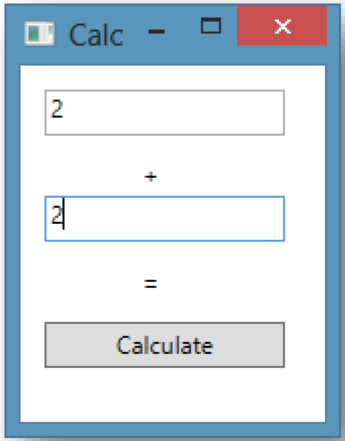


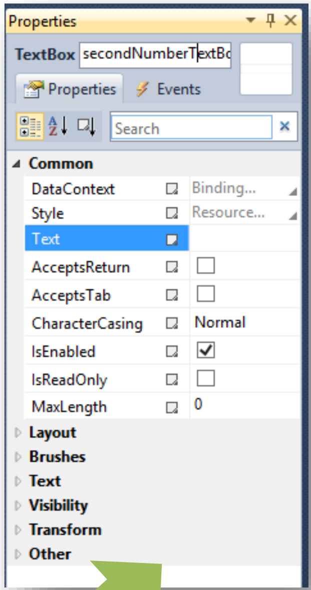
XAML file content

```
<TextBox Height="23" HorizontalAlignment="Left"
Margin="12,12,0,0" Name="firstNumbertextBox"
VerticalAlignment="Top" Width="120" />
```

- This snippet of XAML is the description of a textbox on the screen `firstNumberTextBox` in the AddingMachine application
- It contains fields that describe the position and size of the textbox
- This file is managed by Visual Studio as your program is being developed

XAML in Visual Studio





Properties

TextBox secondNumberTextBo

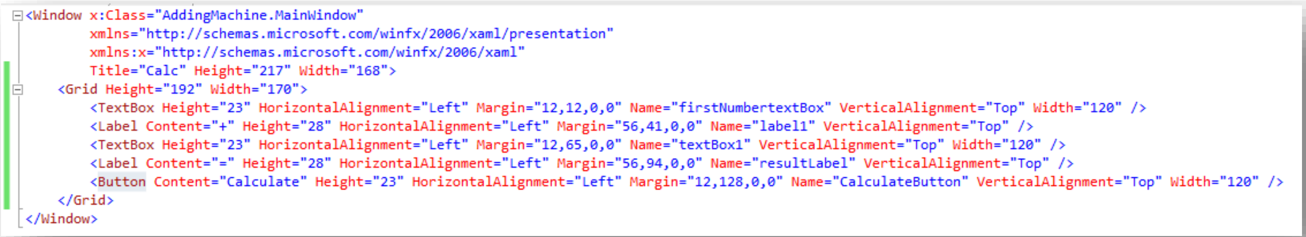
Properties Events

Search

Common

DataContext	Binding...
Style	Resource...
Text	
AcceptsReturn	☐
AcceptsTab	☐
CharacterCasing	Normal
IsEnabled	☑
IsReadOnly	☐
MaxLength	0

Layout
Brushes
Text
Visibility
Transform
Other



```
<Window x:Class="AddingMachine.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Calc" Height="217" Width="168">
    <Grid Height="192" Width="170">
        <TextBox Height="23" HorizontalAlignment="Left" Margin="12,12,0,0" Name="firstNumbertextBox" VerticalAlignment="Top" Width="120" />
        <Label Content="+" Height="28" HorizontalAlignment="Left" Margin="56,41,0,0" Name="label1" VerticalAlignment="Top" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="12,65,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" />
        <Label Content="=" Height="28" HorizontalAlignment="Left" Margin="56,94,0,0" Name="resultLabel" VerticalAlignment="Top" />
        <Button Content="Calculate" Height="23" HorizontalAlignment="Left" Margin="12,128,0,0" Name="CalculateButton" VerticalAlignment="Top" Width="120" />
    </Grid>
</Window>
```

- The XAML file holds the information which is updated by both views

The XAML language

- XAML is a "declarative" language
- It just tells us about things, it does not tell us what they do and how they can do it
- The XAML file has a particular format
  - The characters < and > are used to mark the start and end of some elements in the file
- The format looks a bit like XML
  - eXtensible Markup Language

Using XAML

- You can actually edit the XAML text in your project to create new display elements and modify existing ones
- This can often be much quicker than using the editing interface provided by Visual Studio
- You just have to type the new values into the XAML window and the properties of the element are changed immediately

UNIVERSITY OF Hull

The XAML file at run time

- When a WPF program runs the XAML file is compiled into a set of low level display instructions that are obeyed by the runtime system
- This is the point at which the XAML object descriptions in the text are converted into program objects we can use in our code
- This all happens automatically as far as we are concerned
- The program can just use the display elements as objects in the code, rather like we use the `Console` object

XAML and XML

- XAML looks a bit like XML
  - XML means “Extensible Markup Language”
- This means that XML is really a way of designing languages that want to talk about something
- Just like the english language lets us invent verbs and nouns and put them into sentences that have meaning in a particular context

Quick intro to XML

```xml
<?xml version="1.0" encoding="us-ascii" ?>
<HighScoreRecords count="2">
    <HighScore game="Breakout">
        <playername>Rob Miles</playername>
        <score>1500</score>
    </HighScore>
    <HighScore game="Space Invaders">
        <playername>Rob Miles</playername>
        <score>4500</score>
    </HighScore>
</HighScoreRecords>
```

- I invented this XML format to hold a video game high score table

HighScore element

```
<HighScore game="Breakout">
        <playername>Rob Miles</playername>
        <score>1500</score>
 </HighScore>
```

- The `HighScore` element contains two other elements, `playername` and `score`
- It also has a property that gives the name of the game
- I could add others, for example the date and time the score was achieved
- It is easy for us to work out what the elements are there for

HighScoreRecords element

```
<?xml version="1.0" encoding="us-ascii" ?>
<HighScoreRecords count="2">
    <HighScore game="Breakout">
        <playername>Rob Miles</playername>
        <score>1500</score>
    </HighScore>
    <HighScore game="Space Invaders">
        <playername>Rob Miles</playername>
        <score>4500</score>
    </HighScore>
</HighScoreRecords>
```

- The `HighScoreRecords` element contains a count of the number of `HighScore` elements

XML and data structures

- We can invent our own language format whenever we have some structured data that we want to store
- The designers of XAML have done this
- Rather than store high scores they have created a language that lets us design user interfaces

The XAML data revisited

```
<TextBox Height="23" HorizontalAlignment="Left"
Margin="12,12,0,0" Name="firstNumbertextBox"
VerticalAlignment="Top" Width="120" />
```

- We can see that the XAML content that describes a textbox is very similar to a `HighScore` element
- The designers of XAML had to work out what data fields are required in a `TextBox` object
- Each display element has a set of fields
- Visual Studio provides intellisense to help you create these

What is a Markup Language?

- The "ML" in XML stands for "Markup Language"
- A markup language was originally a set of commands for the printers of a document
  - 'Put the words "Table of Contents" in bold'
- When the World Wide Web was created the Hyper Text Markup Language was designed to allow a text file to describe a particular web page design
- However, there are lots of other markup languages available

XML and HTML

- The idea of creating your own markup language was such a good one that people wanted a standard form for doing this
- XML came out of this drive for standards
  - It is the way in which the files use the < and /> and other characters to mean the start and end of elements, names and properties
  - It also tells you how to create "schemas" that define the structure and content of XML documents

XML Schema

- An XML schema describes a particular XML document format:
  - “A HighScore element must contain a PlayerName and a Score value, but the Date value is optional”
- Programs can use a schema to make sure that a particular document contains content which is valid
- The schema in use is identified in the header of an XML document
- Microsoft have created a schema for the XAML language

XML and software

- XML allows programs to share data irrespective of what kind of system was used to create the data
- There are many software tools that can create schemas and you can even store the contents of C# directly into XML structured files
- However, for now just remember that the description of a WPF page is a text file containing an XAML document which is formatted according to XML using a schema that determines how all the elements are to be used

XAML and WPF Pages

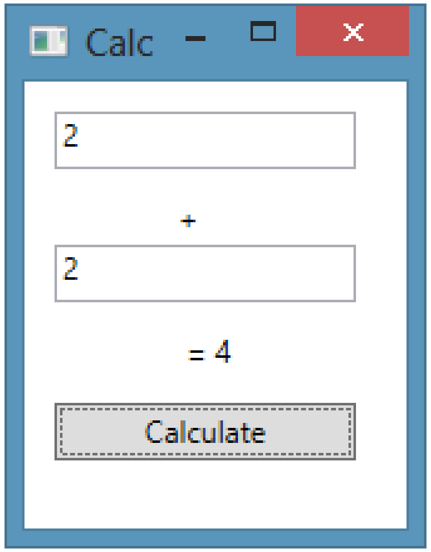
- A WPF application is made up of a number of *pages*
- Each page is expressed using a single XAML source file
- The page will contain descriptions of a number of WPF elements
  - Some elements can contain other elements
- Visual Studio manages the XAML source file as we work on the application
- Items described in the XAML appear as objects in the programs that we create

WPF Components

- There are lots of different components available to be added to a window
  - `Label`: a text label
  - `TextBox`: a box the user can type into
  - `Button`: a button that the user can press
- A program can interact with a component by using the behaviours that it provides
  - We can change the text in a `Label` to display a message
  - We can read the text from a `TextBox` to get user input
  - A `Button` can generate events when it is clicked

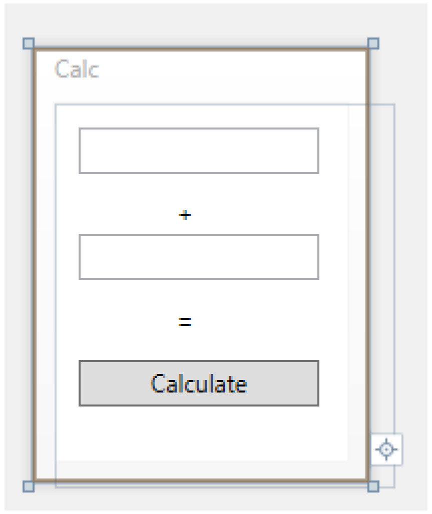
Sample Application

- The Calc program just adds two numbers together
- The user enters the numbers and presses the button to start a calculation
- The result is displayed using a label



Designer View

- This is the designer view of the application
- I added each item in turn to the screen
  - Visual Studio provides some very good tools to help line the items up
- I can also change the size of the application window by dragging the handles attached to the window



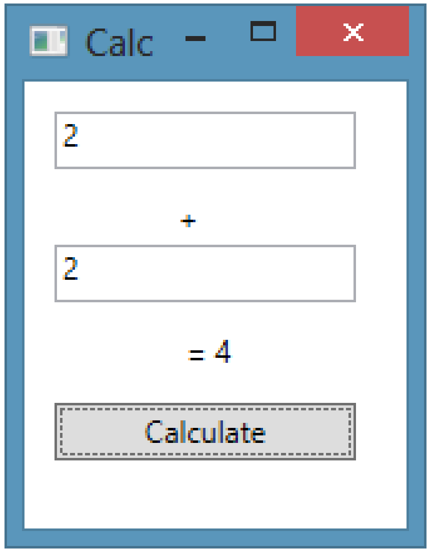
XAML View

```
<Grid Height="192" Width="170">
     <TextBox Height="23" HorizontalAlignment="Left“
       Margin="12,12,0,0" Name="firstNumbertextBox"
       VerticalAlignment="Top" Width="120" />
     <Label Content="+" Height="28" HorizontalAlignment="Left"
       Margin="56,41,0,0" Name="plusLabel" VerticalAlignment="Top" />
     <TextBox Height="23" HorizontalAlignment="Left"
       Margin="12,65,0,0" Name="secondNumberTextBox"
       VerticalAlignment="Top" Width="120" />
     <Label Content="=" Height="28" HorizontalAlignment="Left"
       Margin="56,94,0,0" Name="resultLabel" VerticalAlignment="Top" />
     <Button Content="Calculate" Height="23" HorizontalAlignment="Left"
      Margin="12,128,0,0" Name="CalculateButton" VerticalAlignment="Top"
      Width="120" />
</Grid>
```

- This is the XAML that describes the items in the window

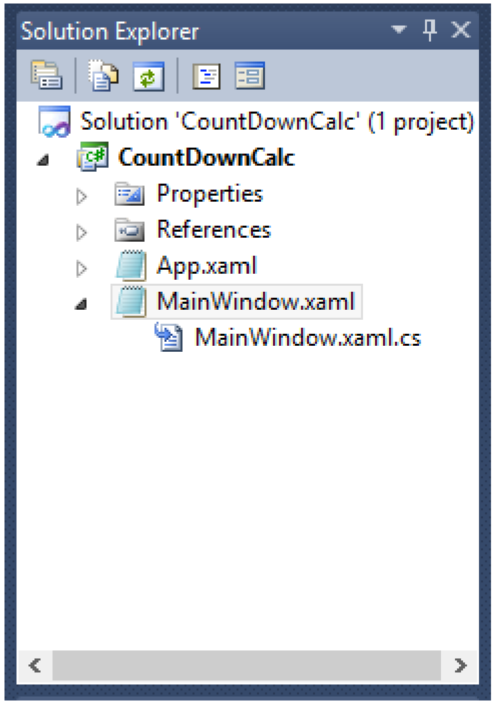
Buttons and Events

- The `CalculateButton` component will appear on the form and the user can click it
- However, at the moment the button doesn't do anything
- What we need to do next is bind an *event* to button
- In other words, we want some C# to run when the button clicked



XAML designs and C# Code

- Each XAML page has a C# program page which is shown in Solution Explorer as being “behind” the window
- Each Window in an application is implemented by a class
- This is where a programmer can put code that makes the user interface work
- This includes the handler for the button clicked event



An Empty Window Class

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

- An empty window just contains a call to the `InitializeComponent` method
- This call is made when the constructor for the window is called
- The method creates all the components that appear on the screen

Window Class Methods

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

- When we add code that responds to events from the user we will put this code into the `MainWindow` class
- The methods that respond to button press events run in here
- The methods that display values to the user will run in here

Responding to Events

- When `CalculateButton` is clicked it needs a way of "telling" a program that this event has occurred
- In C# an event is delivered by a call of a method
  - Our program will contain a `calculateButton_Click` method that is called when the finish button is clicked
  - This will read the new text back from the `TextField` and update the name of our customer
- We need a way of connecting the `CalculateButton` component to the method we want it to call when it is clicked

Referring to Methods using Delegates

- We are familiar with the use of references to refer to objects
  - A reference is a tag that can be tied to a particular object in memory
- Delegates are an extension of references which refer to methods rather than objects
  - The value of a delegate can be set to refer to a method in a class
- We can connect buttons to methods by doing this:
  - Create a delegate that refers to the method we want to use
  - Give this delegate to `calculateButton` so that it knows who to call when the button is clicked

Connecting to the Component

```
<Button Content="Calculate" Height="23"
HorizontalAlignment="Left" Margin="12,128,0,0"
Name="CalculateButton" VerticalAlignment="Top"
Width="120" Click="CalculateButton_Click" />
```

- The XAML that describes the button can contain a `Click` value that identifies the method to be called when the button is clicked
- Visual Studio will do the "plumbing" behind the scenes to create the method and the delegate and connect it all to the button
- We will discover how this works later in the course

Creating the Event Handler

- The simplest way to create an event handler for button is to double click on the button in the Visual Studio graphical user interface
- This will update the XAML as shown above and create an event handler in the window class that we can add code to
- You can also manage the events that a component produces by managing its properties
- Each component can generate a particular set of events

The Event Handler in a Window Class

```
public partial class MainWindow : Window
{
    private void CalculateButton_Click(object sender,
                                       RoutedEventArgs e)
    {
    }
}
```

- This is the empty event handler
- Our program can ignore the parameters (although these can be used so it can determine which object generated the event)
- The method is called each time the button is clicked by the user

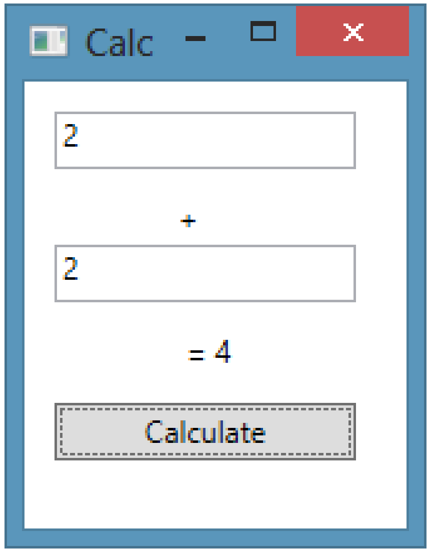
Performing the Calculation

```
private void CalculateButton_Click(object sender,
                                   RoutedEventArgs e)
{
    int v1 = int.Parse(firstNumbertextBox.Text);
    int v2 = int.Parse(secondNumberTextBox.Text);

    int result = v1 + v2;

    resultLabel.Content = " = " + result;
}
```

- Visual Studio makes an empty method for the event handler
- We can fill in the code to make it perform the required task
- In this case it calculates the result and displays it

Running the Program

- The interesting thing about this program is that once it has loaded the window onto the screen it then does nothing
  - There is a Main method in the application, but this just starts off creating the window
- Once the program is active it is simply waiting for the user to press the calculate button



IMPORTANT MESSAGE

- A Window is just the thing that displays the user interface for your program
  - It provides a link between the user and the data objects that they are working with

You should not try to store any of your business data inside the Window class

Sensible Way To Work

```
public partial class MainWindow : Window
{
    Bank activeBank;
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

- The variable `activeBank` contains a reference to the bank that the user is working with
- The bank will contain methods that will let code in the user interface find accounts and get data from them for display

Stupid Way To Work

```
public partial class Account : Window
{
    string customerName;
    public Account()
    {
        InitializeComponent();
    }
}
```

- The program is trying to store business data (the name of a customer) inside the Window class that is driving the user interface
- This is not the right thing to do, we don't want to have to store buttons and labels when we store a customer

Very Sensible Way To Work

```
public partial class CustomerEditWindow : Window
{
    string selectdCustomerName;
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

- This is **much** more sensible
- The string is set to the name of the customer account that is currently being edited
- Methods in the window could update this name and save it back in the account

Windows Presentation Foundation Summary

- Windows are displays on the screen that are manipulated as C# objects
- The design of the objects on the screen is expressed using the XAML language
- Windows can contain components such as `Label`, `TextBox` and `Button`
- The `Button` component can generate an event when it is clicked
- You can use delegates to tell the button which method to call when a click event occurs