Windows® Phone

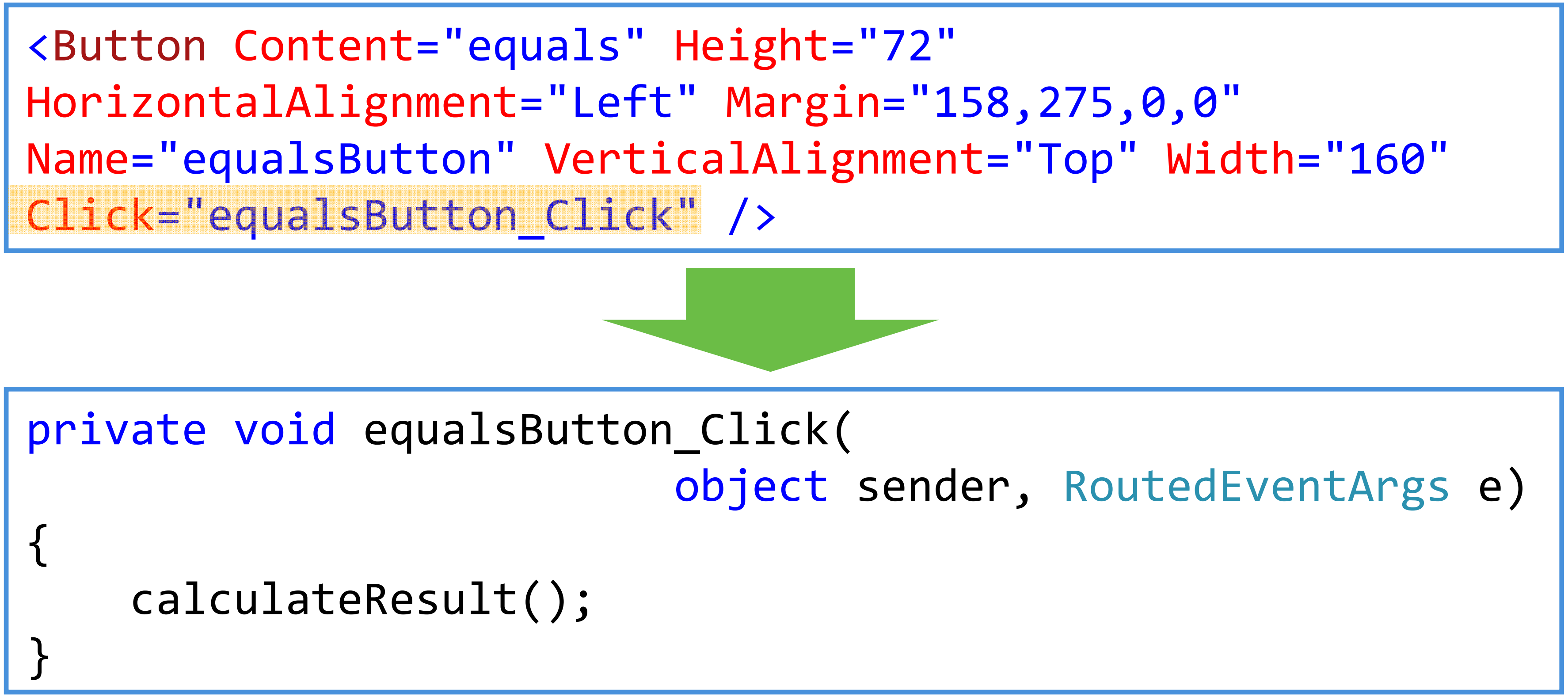
Data Manipulation and Display

Session 4.2

Topics

- Responding to events from Silverlight elements
- Using DataBinding to connect Silverlight elements to application classes
  - Adding classes as resources to a page
  - One way data binding
  - Two way data binding

Silverlight element events



```
<Button Content="equals" Height="72"
HorizontalAlignment="Left" Margin="158,275,0,0"
Name="equalsButton" VerticalAlignment="Top" Width="160"
Click="equalsButton_Click" />
```

```
private void equalsButton_Click(
                         object sender, RoutedEventArgs e)
{
    calculateResult();
}
```

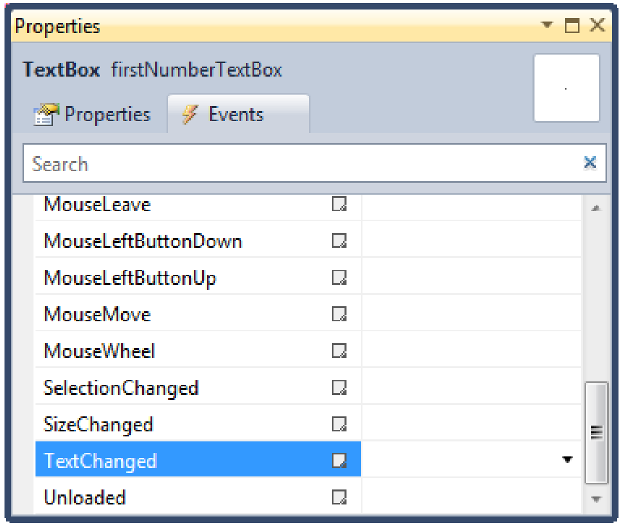
- We have seen how Silverlight elements can link to event handlers in the code for a page

The TextChanged Event

- A `Button` can generate `Click` events when it is clicked
- A `TextBox` can generate `TextChanged` events if the user types in the `TextBox`
- We can use this to create an `AddingMachine` that updates automatically
- No need to press `Calculate` each time a new solution is required

TextBox Events

- The **TextBox** can generate lots of different events
- We can create a binding to an event by double clicking the event in the **Properties** pane
- This updates the xaml description of the control and adds an event handler



MouseLeave
MouseLeftButtonDown
MouseLeftButtonUp
MouseMove
MouseWheel
SelectionChanged
SizeChanged
TextChanged
Unloaded

Automatically calculating a result

```
private void firstNumberTextBox_TextChanged(
                    object sender,TextChangedEventArgs e)
{
     calculateResult();
}
```

- Each time the text in the first number **TextBox** changes the calculation is performed
- We can also bind an event to changes in the second **TextBox** too

Double TextChanged Events

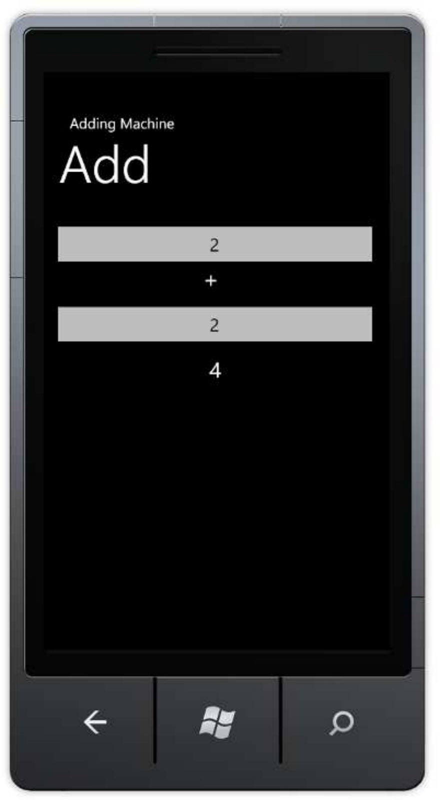
```
string oldFirstNumber = "";
private void firstNumberTextBox_TextChanged(
             object sender, TextChangedEventArgs e)
{
    if (firstNumberTextBox.Text == oldFirstNumber)
return;
    oldFirstNumber = firstNumberTextBox.Text;

    calculateResult();
}
```

- There is a known issue with the **TextChanged** event. It may get fired twice
- This code shows how to work round this

Demo

Demo 1: Auto update AddingMachine



Adding Machine

Add

2

+

2

4

Data Binding

- This is a **very important** subject
- It provides a coupling between the data in a program and the user interface
- It removes the need to write “glue” logic to link program data to display elements
- It can work in two ways
  - ‘One-way’
  - ‘Two-way’

One Way Data Binding

- This creates a connection between a property in a display object and a property in a C# class
- If the property in the class is changed, the property in the display element is changed too
- We can use this in our **`AddingMachine`** to bind the result of the calculation to the display on the page

Two Way Data Binding

- This form of connection works in both directions
    - Changes to the display element fire events in the C# class
    - Changes to properties in the C# class cause the display element to update
- We can use this in our **`AddingMachine`** to read numbers in from the user

Creating an Adder class

- The first thing we need to do is create a class that encapsulates the behaviour of the `AddingMachine`
- This will expose the properties that we can bind to the display elements
  - Text in the top `TextBox`
  - Text in the bottom `TextBox`
  - Text in the result `TextBlock`

Creating an object to bind to

```
public class AdderClass
{
    private int topValue;
    public int TopValue
    {
        get { return topValue; }
        set { topValue = value; }
    }
    // repeat for bottomValue

    public int AnswerValue
    {
        get { return topValue + bottomValue;}
    }
}
```

The AdderClass

- The AdderClass is the element in our program that will do all the business logic
- We could give it to another programmer to fill in and they could put the behaviours behind the properties in the class
  - We could also create test versions to test the display
- This is another example of separation of the program from the presentation

Adding Notification Behaviour

```
public interface INotifyPropertyChanged
{
    // Summary:
    //     Occurs when a property value changes.
    event PropertyChangedEventHandler PropertyChanged;
}
```

- For a class to be able to bind to display elements it must implement the **INotifyPropertyChanged** interface
- This is how elements can register an interest in properties in the class

Silverlight display elements

```
PropertyChanged(this,
                new PropertyChangedEventArgs("AnswerValue"));
```

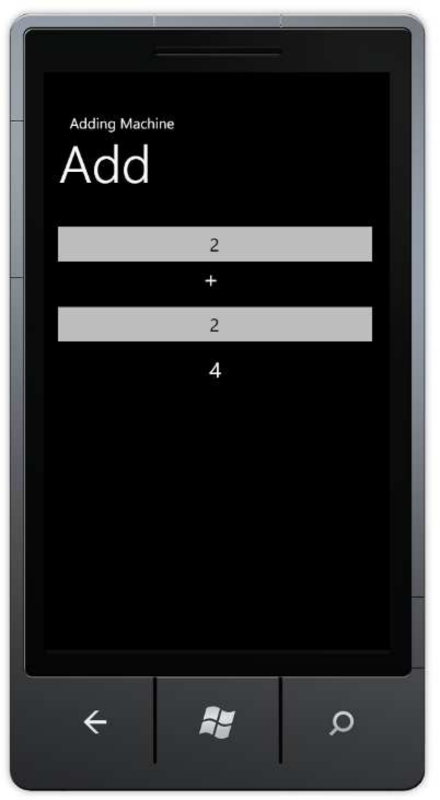
- When a Silverlight display element wants to be notified of a change to a property it will bind to the **PropertyChanged** event
- The data object can then use the event to deliver a message that a property has changed
- Above we are telling Silverlight the **AnswerValue** has changed

Reading back the property

```
public int AnswerValue
{
    get
    {
        return topValue + bottomValue;
    }
}
```

- When the Silverlight element reads the property it causes the result to be calculated and returned
- This value will end up on the display

Demo


Adding Machine

Add

2

+

2

4

Demo 2: AddingMachine with data binding

Binding a class to the xaml

```
xmlns:local="clr-namespace:AddingMachine"
```

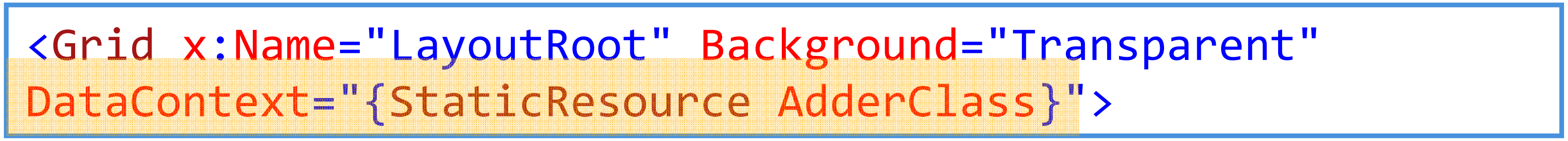
- The C# business class must be bound to the xaml in a page so that the page can use it
- We start by adding the namespace containing the class to the resources available to this page
- A xaml file contains a list of the namespaces it is using

Mapping a class to a resource

```
<phone:PhoneApplicationPage.Resources>
    <local:AdderClass x:Key="AdderClass" />
</phone:PhoneApplicationPage.Resources>
```

- The next step is to make a mapping between the **`AdderClass`** and the name we are using in the xaml page
  - In this case we are using the same name for both
  - You might want to use different classes or test classes in the xaml page
  - This lets you configure this in the xaml

Adding the resource to an element



```
<Grid x:Name="LayoutRoot" Background="Transparent"
DataContext="{StaticResource AdderClass}">
```

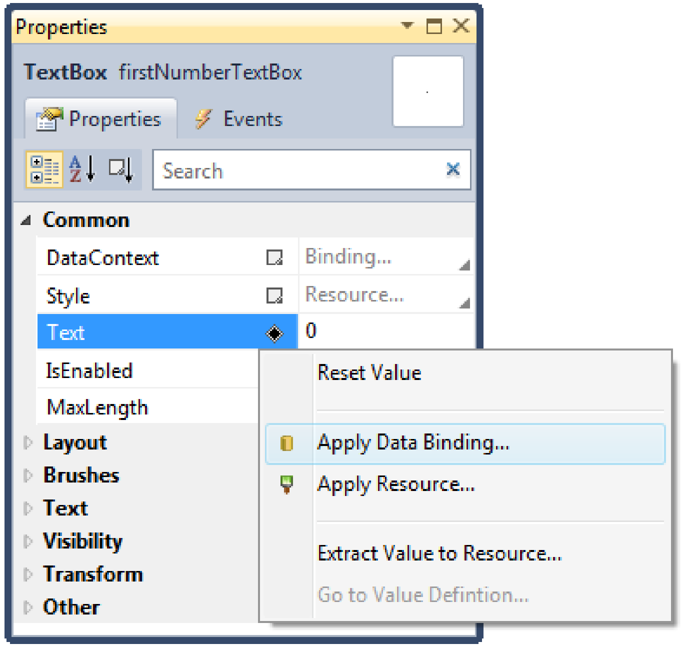
- The **Grid** control is the one that holds all the elements on our page
- Adding a **DataContext** to a control makes it available to all the controls it contains
- This means that our **TextBlock** and **TextBox** items can now all use that control

Business Objects and Silverlight

- We have now connected our **`AdderClass`** business object to **`Grid`** element and all the components in it
- We only have to do this once for a class, no matter how many properties it contains
- The next thing we need to do is bind the Silverlight element properties to the properties exposed by the **`AdderClass`** object

Binding the top line to AdderClass

- We are going to connect the **Text** in the first number textbox (the one in the top line) to the **TopValue** property in **AdderClass**
- We do this from the **Properties** pane in Visual Studio



TextBox firstNumberTextBox

DataContext Binding...

Style Resource...

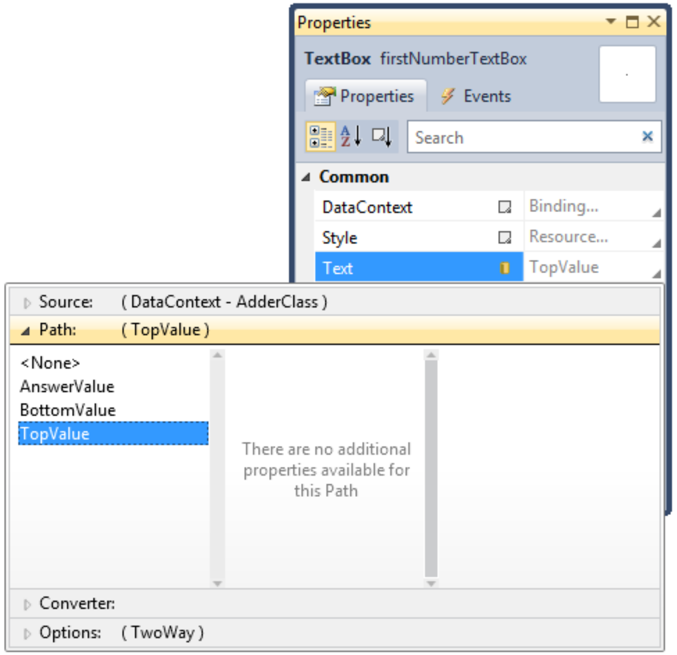
Text 0

IsEnabled

MaxLength

Applying Data Binding

- If we select "**Apply Data Binding**" we are given a list of data sources and properties they expose
- We select the one we want to use
- Note that we have selected **TwoWay** binding



TextBox firstNumberTextBox

DataContext Binding...

Style Resource...

Text TopValue

Source: (DataContext - AdderClass)

Path: (TopValue)

<None>
AnswerValue
BottomValue
TopValue

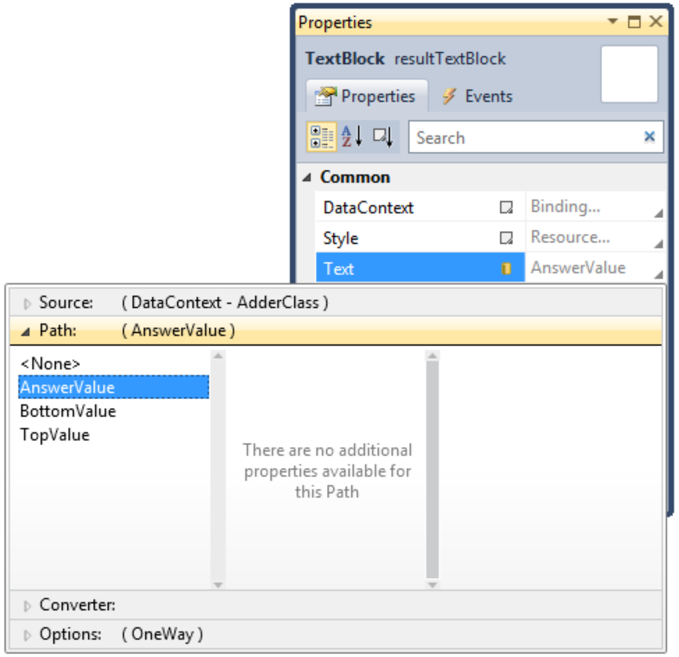
There are no additional properties available for this Path

Converter:

Options: (TwoWay)

One Way binding for the answer

- When we bind the result text box this is only bound "OneWay"
  - There is no set method for this property in `AdderClass`
  - The program will only ever want to display results
- The binding can only be `One Way`



Properties

TextBlock resultTextBlock

DataContext: Binding...

Style: Resource...

Text: AnswerValue

Source: (DataContext - AdderClass)

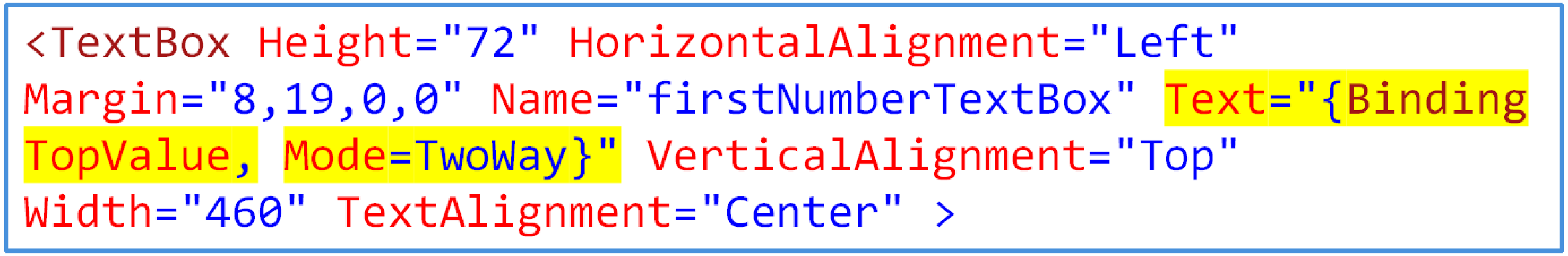
Path: (AnswerValue)

There are no additional properties available for this Path

Converter:

Options: (OneWay)

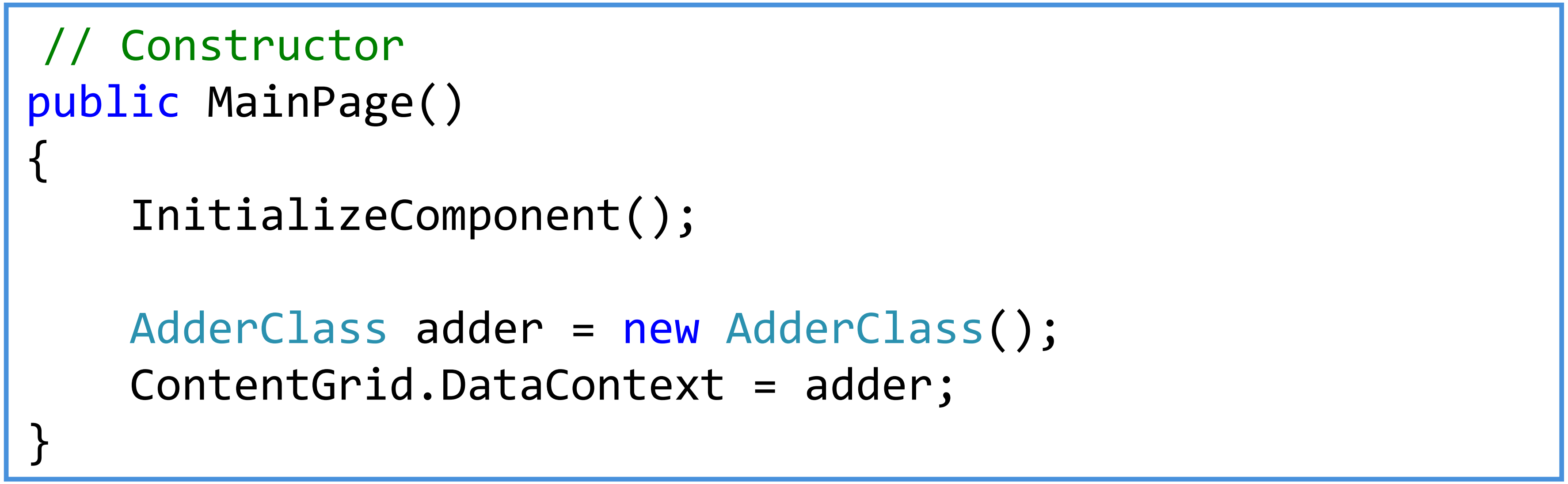
Databinding and the DataContext



```
<TextBox Height="72" HorizontalAlignment="Left"
Margin="8,19,0,0" Name="firstNumberTextBox" Text="{Binding
TopValue, Mode=TwoWay}" VerticalAlignment="Top"
Width="460" TextAlignment="Center" >
```

- It turns out that there is a much easier way of binding data to elements on a page
- We can set a simpler form of the Binding value to identify a property, and then set the DataContext of the element in our program

Databinding and the DataContext



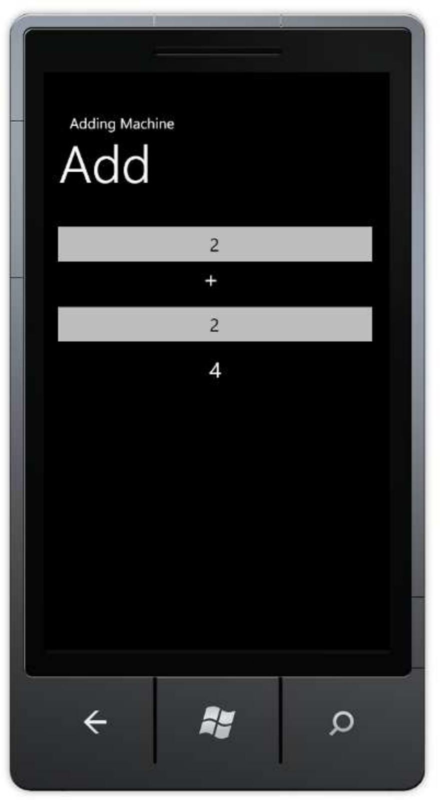
```
 // Constructor
public MainPage()
{
    InitializeComponent();

    AdderClass adder = new AdderClass();
    ContentGrid.DataContext = adder;
}
```

- The ContentGrid is the container for our display elements
- Setting the DataContext to an AddClass instance binds element properties to it

Demo

Demo 3: AddingMachine with data context



Review

- Silverlight elements can generate events that C# code can bind to
  - This includes changing text in a TextBox
- Silverlight elements properties can also be bound to properties in C# business objects
  - The object implements an interface that exposes an event source that Silverlight can bind to
- This can provide two way (input/output) or one way (output only) data binding