

Controls in WPF



Agenda

- What Are Controls?
- Control Architecture
- Introduction to some common controls

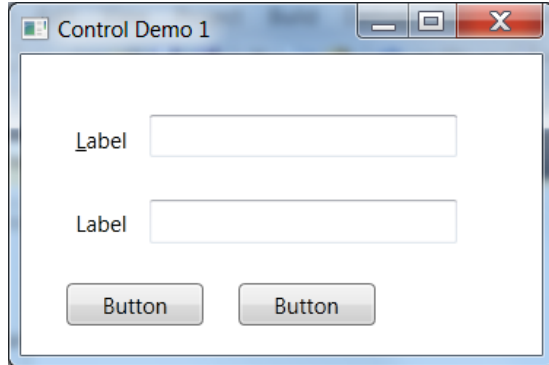
What Are Controls?

- Controls in WPF are all about behavior
 - and they defer to templates to provide their visuals
- Customizing a control's appearance is simple
 - by use of nested content and templates
- You only need to write a custom control when you need interactive behavior that is different from any of the built-in controls
- Not all WPF user interface elements are controls.
 - Shapes like Rectangle and Ellipse have no intrinsic behavior
 - they are just about appearance
 - Lower-level elements do not derive from Control
 - Usually they derive from FrameworkElement

Programming Model

- The three principles of controls:

- **Element composition**



```
<XAML>
<Grid Height="174" Width="293">
  <Button Height="23" HorizontalAlignment="Left"
  <Button Height="23" Margin="119,0,99,26" Name="button1"
  <TextBox Height="23" Margin="70,33,54,0" Name="textBox1"
  <TextBox Height="23" Margin="70,79,54,71" Name="textBox2"
  <Label Height="23" Margin="25,33,0,0" Name="label1"
  <Label Height="23" Margin="25,79,0,71" Name="label2" HorizontalAlignment="Left"
</Grid>
```

- **Rich content**



```
Button b = new Button();
StackPanel sp = new StackPanel();
b.Content = sp;
Ellipse left = new Ellipse();
TextBlock text = new TextBlock();
text.Inlines.Add(new Run("Hello "));
text.Inlines.Add(new Bold(new Run("World")));
sp.Children.Add(left);
sp.Children.Add(text);
```

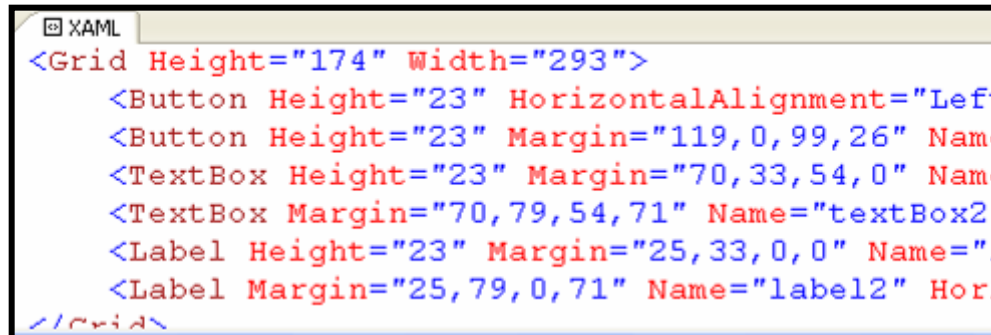
- **Simple programming model**



```
Button sb = new Button();
sb.Content = "Click Me";
sb.Click += sb_Click;
```

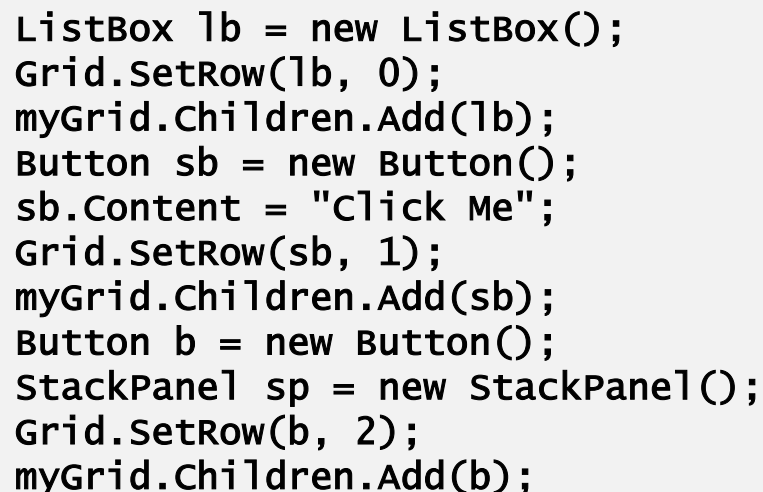
XAML or C#

- Composition of controls can be done in:
 - XAML

A screenshot of a text editor window showing XAML code. The code defines a Grid with a height of 174 and width of 293. Inside the Grid, there are two Buttons, two TextBoxes, and two Labels, each with specific height, margin, and name attributes. The code is color-coded: tags are blue, attributes are red, and values are black.

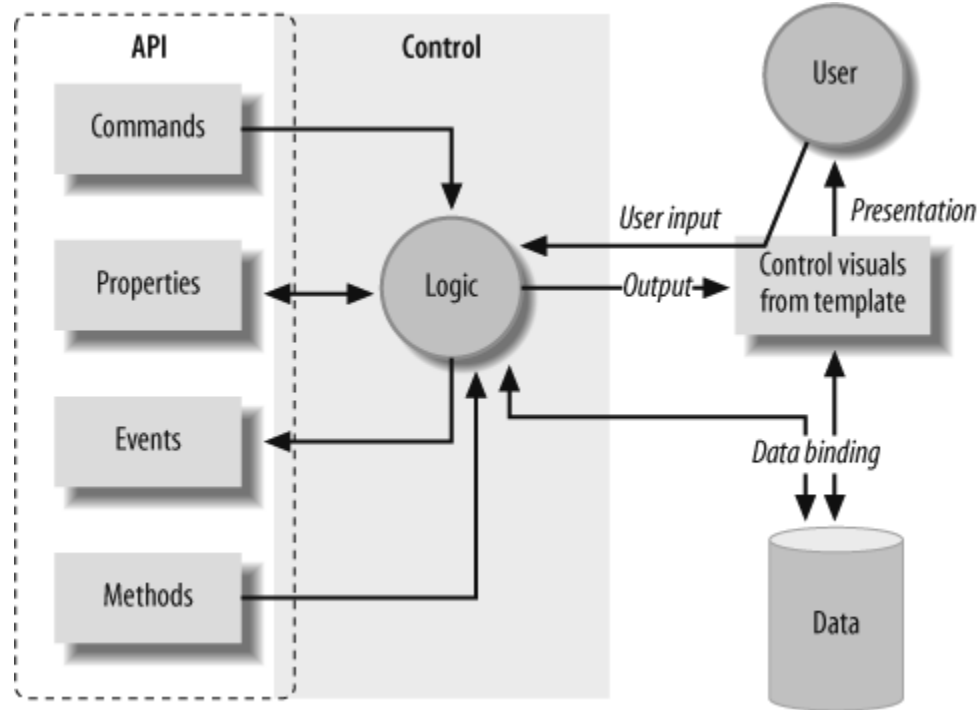
```
<? XAML
<Grid Height="174" Width="293">
    <Button Height="23" HorizontalAlignment="Left
    <Button Height="23" Margin="119,0,99,26" Name
    <TextBox Height="23" Margin="70,33,54,0" Name
    <TextBox Margin="70,79,54,71" Name="textBox2"
    <Label Height="23" Margin="25,33,0,0" Name="l
    <Label Margin="25,79,0,71" Name="label2" Hor
</Grid>
```

- Or in code (e.g. C#)

A screenshot of a text editor window showing C# code. The code creates a ListBox, a Button, and a StackPanel, and adds them to a Grid. The code is color-coded: keywords are blue, variables and property names are black, and string literals are red.

```
ListBox lb = new ListBox();
Grid.SetRow(lb, 0);
myGrid.Children.Add(lb);
Button sb = new Button();
sb.Content = "Click Me";
Grid.SetRow(sb, 1);
myGrid.Children.Add(sb);
Button b = new Button();
StackPanel sp = new StackPanel();
Grid.SetRow(b, 2);
myGrid.Children.Add(b);
```

Control Architecture



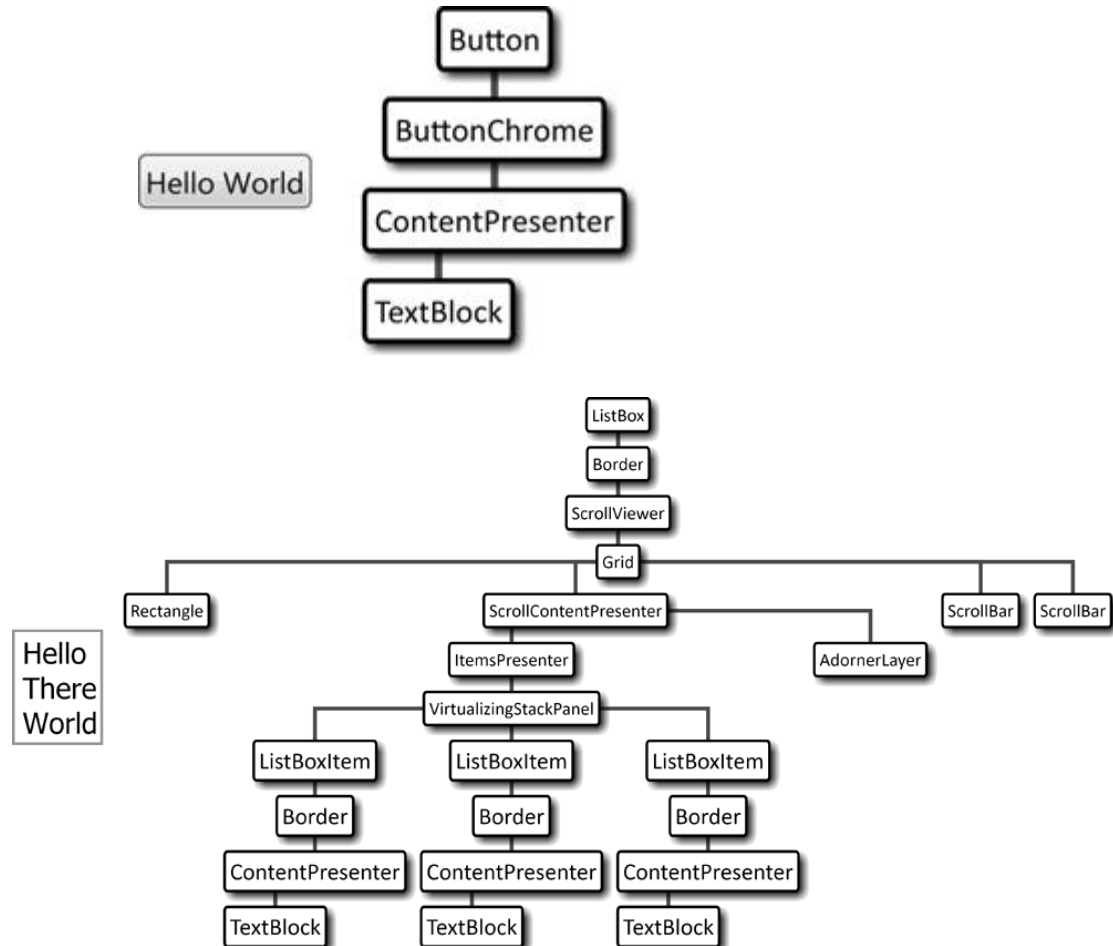
- Controls in WPF uses a variation of the famous **Model View Controller** design pattern
 - The **Model** is (can be) attached to the Control by use of data binding
 - Model is a name for objects representing the underlying data
 - The template can be seen as the **View**
 - Contains the objects that display that data
 - And the control plays the role of the **Controller**
 - Contains objects that manage input from the user and interactions between the model and the view

The Display Tree

- The hierarchy of elements, also known as the **display tree**, created to display the button:

```
Button b = new Button();  
b.Content = "Hello World";
```

```
ListBox l = new ListBox();  
l.Items.Add("Hello");  
l.Items.Add("There");  
l.Items.Add("World");
```

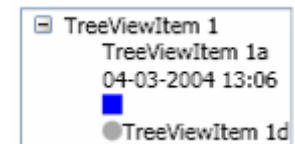


ContentPresenter Algorithm

1. If Content is of type UIElement, then add it to the display tree
2. If ContentTemplate is set, use that to create a UIElement instance and add it to the display tree
3. If ContentTemplateSelector is set, use that to find a template, use the template to create a UIElement instance, and add it to the display tree
4. If the data type of Content has a data template associated with it, use that to create a UIElement instance
5. If the data type of Content has a TypeConverter instance associated with it that can convert to type UIElement, convert Content and add it to the display tree
6. If the data type of Content has a TypeConverter instance associated with it that can convert to a string, wrap Content in TextBlock and add it to the display tree
7. Finally, call ToString on Content, wrap it in TextBlock, and add it to the display tree

The Content Model

- The content model specifies the types of objects that a control can contain
- There are four different content models:
 - **ContentControl**
 - contains a single item
 - has a **Content** property of type Object
 - **HeaderedContentControl**
 - contains a header and a single item
 - inherits the **Content** property from ContentControl and defines the **Header** property that is of type Object
 - **ItemsControl**
 - contains a collection of items
 - you can use either the **ItemsSource** property or the **Items** property to populate an ItemsControl
 - **HeaderedItemsControl**
 - contains a header and a collection of items
 - inherits from the ItemsControl class, and adds the **Header** property that is of type Object
- Some controls have their own special content model
 - E.g. a TextBox has the content property **Text**, and can only contain unformatted text



Introduction to some common controls

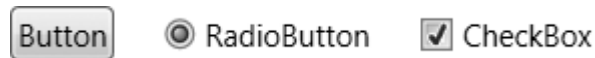
ContentControls

- The following controls use the ContentControl content model:
 - Button
 - ButtonBase
 - CheckBox
 - ComboBoxItem
 - ContentControl
 - Frame
 - GridViewColumnHeader
 - GroupItem
 - Label
 - ListBoxItem
 - ListViewItem
 - NavigationWindow
 - RadioButton
 - RepeatButton
 - ScrollViewer
 - StatusBarItem
 - ToggleButton
 - ToolTip
 - UserControl
 - Window
- You can use either XAML or code to set the Content of the control
- There are no restrictions on what you can put in a ContentControl

Buttons

- Buttons are controls that a user can click
- The result of the click is up to the application developer
 - but there are common expectations depending on the type of button
 - clicking on a CheckBox or RadioButton expresses a choice, and does not normally have any immediate effect beyond visually reflecting that choice
 - clicking on a normal Button usually has some immediate effect

- Button types:



- Buttons with nested content:



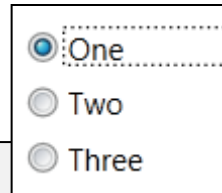
- Button with access key:



Use of RadioButtons

- Ordinarily, radio buttons are grouped by their container
 - If you place three RadioButton controls in a single StackPanel, they form a group from which you can select just one of the three

```
<StackPanel>  
  <RadioButton Checked="rbChecked">One</RadioButton>  
  <RadioButton Checked="rbChecked">Two</RadioButton>  
  <RadioButton Checked="rbChecked">Three</RadioButton>  
</StackPanel>
```



- The GroupName property allows you to override this behavior
 - You can use it to create more than one group in the same container or to create a single group that spans multiple containers

Label

- Label is used to provide a caption for controls that do not have their own built-in caption
 - The most widely used example is the TextBox control
- The purpose of the Label control is to provide a place to put a caption with an access key
 - When the access key is pressed, the Label will redirect the focus to the relevant control

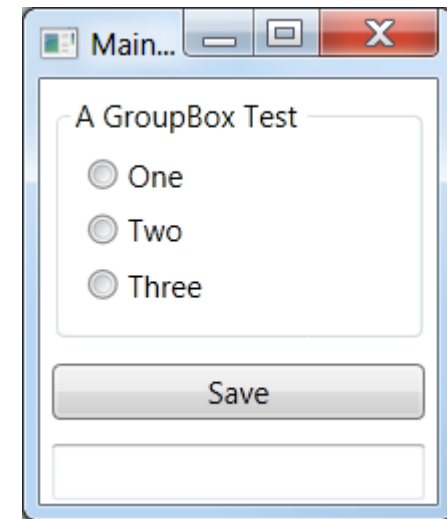
Name: Quest:

```
<Label Target="{Binding ElementName=nameText}">_Name:</Label>  
<TextBox x:Name="nameText" width="70" />  
<Label Target="{Binding ElementName=questText}">_Quest:</Label>  
<TextBox x:Name="questText" width="70" />
```

Headed Content Controls

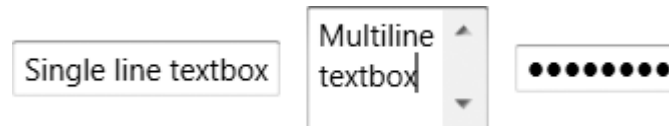
- Three classes derive from HeaderedContentControl:
 - GroupBox
 - TabItem
 - Expander
- The GroupBox is displayed as a box with rounded corners and a title

```
<GroupBox Header="A GroupBox Test"
          Padding="5"
          Margin="5"
          VerticalAlignment="Top"
          >
  <StackPanel>
    <RadioButton Name="rb1"
    ...
```



Text Controls

- Have their own special content model
- The simplest text editing control is TextBox
 - By default, it allows a single line of text to be edited
 - By setting `AcceptsReturn` to true, and the `TextWrapping` property to `Wrap` it can edit multiple lines
 - It provides basic text editing facilities: selection support, system clipboard integration (cut, paste, etc.), and multilevel undo support
- TextBox and PasswordBox



- TextBox support only plain text
- TextBox provides a `Text` property that represents the control's contents as a `String`
- A `RichTextBox` edits a `FlowDocument`, which can contain a wide variety of content:

***This** is a Rich**Text**Box.*

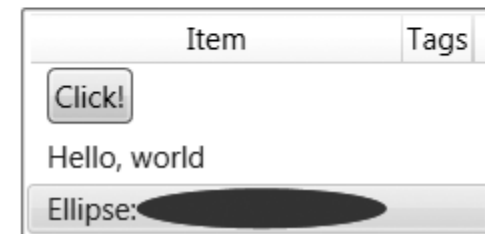
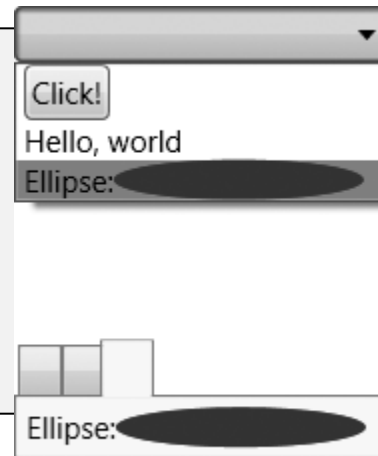
ItemsControls

- The following controls use the ItemsControls content model:
 - ComboBox
 - ContextMenu
 - ListBox
 - ListView
 - Menu
 - StatusBar
 - TabControl
 - TreeView

Typical List Controls

- **ListBox**, **ComboBox**, **ListView**, and **DataGrid** can all present a linear sequence of items
- **TreeView** presents a hierarchy of items
- The **TabControl** may not seem like an obvious relative of the **ListBox**, but it shares the basic features:
 - it presents a sequence of items (tab pages) and lets the user choose which is the current item.

```
<ComboBox>  
  <Button>Click!</Button>  
  <TextBlock>Hello, world</TextBlock>  
  <StackPanel Orientation="Horizontal">  
    <TextBlock>Ellipse:</TextBlock>  
    <Ellipse Fill="Blue" width="100" />  
  </StackPanel>  
</ComboBox>
```



ItemsControls Items

- You can use either the `ItemsSource` property or the `Items` property to populate an `ItemsControl`
- **Items Property**
 - you can add items by using the `Items` property
 - The items in an `ItemsControl` can have types that are different from each other

```
listBox1.Items.Add("This is a string in a ListBox");  
DateTime dateTime1 = new DateTime(2004, 3, 4, 13, 6, 55);  
listBox1.Items.Add(dateTime1);
```

- **ItemsSource Property**
 - enables you to use any type that implements `IEnumerable` as the content of the `ItemsControl`
 - is typically used to display a data collection

```
List<string> myCollection = new List<string>();  
myCollection.Add("item 1");  
listBox1.ItemsSource = myCollection;
```

Prefer

ItemsSource Auto Refresh

- You can use the `ItemsSource` property to bind an `ItemControl` to any kind of collection
- But when you add or remove an item to the collection the `ItemControl` will only update the view if the collection class implements the `INotifyCollectionChanged` interface
 - And only an **ObservableCollection** does that!
- If you bind to any other kind of collection you must manually call `Refresh` to update (redraw) the control

```
public MainWindow()
{
    InitializeComponent();
    myItems = new List<string>();
    listBox1.ItemsSource = myItems;
}

private void btnAdd_Click(...)
{
    myItems.Add("New item ");
    listBox1.Items.Refresh();
}
```

```
public MainWindow()
{
    InitializeComponent();
    myItems = new ObservableCollection<string>();
    listBox1.ItemsSource = myItems;
}

private void btnAdd_Click(...)
{
    myItems.Add("New item ");
}
```

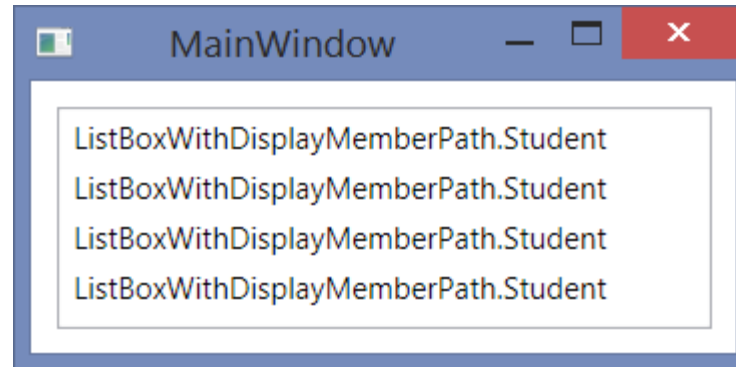
Prefer

DisplayMemberPath

- When you display a custom class in a itemsControl:

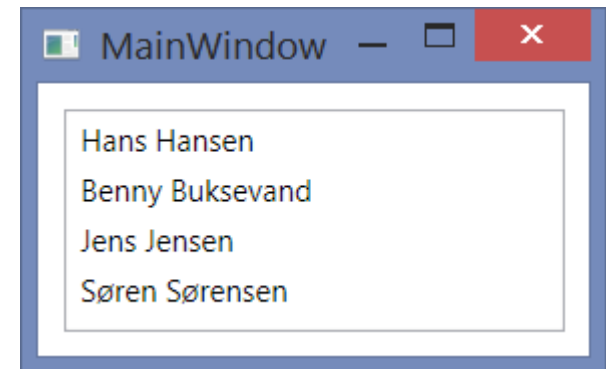
```
lbxStudents.ItemsSource = new ObservableCollection<Student>();
```

you may see this:



- To display some meaningful data you can either:

- Use DisplayMemberPath
- Define a suitable DataTemplate
- Or override ToString()



Item Container Classes

- Each ItemsControl has a corresponding class that represents an item in the ItemsControl
- You can explicitly create an item container for each item in the ItemsControl
 - but it is not necessary

ItemsControl	Item container
ComboBox	ComboBoxItem
ContextMenu	MenuItem
ListBox	ListBoxItem
ListView	ListViewItem
Menu	MenuItem
StatusBar	StatusBarItem
TabControl	TabItem
TreeView	TreeViewItem

HeaderedItemsControl

WPF has 3 HeaderedItemsControl:

- MenuItem
- ToolBar
- TreeViewItem