

CONTROLS

Chapter 6 of Pro WPF : By Matthew MacDonald

Assist Lect. Wadhah R. Baiee . College
of IT – Univ. of Babylon - 2014

Introduction

- *Controls* elements that derive from the `System.Windows.Control` class. You'll begin by examining the base `Control` class, and learning how it supports brushes and fonts. Then you'll explore the full catalog of WPF controls, including the following:

Introduction

- **Content controls:** These controls can contain nested elements. the [Label](#), [Button](#), [ToolTip](#), and [ScrollViewer](#) classes.
- **Headered content controls:** These are content controls that allow you to add a main section of content and a separate title portion. They include the [TabItem](#), [GroupBox](#), and [Expander](#) classes.
- **Text controls:** This is the small set of controls that allow users to enter input. (the [TextBox](#)), (the [PasswordBox](#)), and (the [RichTextBox](#))

Introduction

- **List controls:** These controls show collections of items in a list. They include the [ListBox](#) and [ComboBox](#) classes.
- **Range-based controls:** These controls have just one thing in common: a Value property that can be set to any number in a prescribed range. Examples include the [Slider](#) and [ProgressBar](#) classes.
- **Date controls:** This category includes two controls that allow users to select dates: the [Calendar](#) and [DatePicker](#).

Setting Colors in XAML

- When you set the background or foreground in XAML, you can use a helpful shortcut. The WPF parser will automatically create a [SolidColorBrush](#) object using the color you specify.

```
<Button Background="Red">A Button</Button>
```

It's equivalent to this more verbose syntax:

```
<Button>A Button  
  <Button.Background>  
    <SolidColorBrush Color="Red" />  
  </Button.Background>  
</Button>
```

Fonts

- The Control class defines a small set of font-related properties that determine how text appears in a control.

- [FontFamily](#)
- [FontSize](#)
- [FontStyle](#)
- [FontWeight](#)
- [FontStretch](#)

Font Family

- A *font family* is a collection of related typefaces.
- For example, Arial Regular, Arial Bold, Arial Italic, and Arial Bold Italic are all part of the Arial font family.
- **Example** : the following markup sets the FontFamily to Times New Roman and sets the FontWeight to FontWeights.Bold:

```
<Button FontFamily="Times New Roman" FontWeight="Bold">A Button</Button>
```

Mouse Cursors

- A common task in any application is to adjust the mouse cursor to show when the application is busy or to indicate how different controls work.
- You can set the mouse pointer for any element by using the Cursor property
- If you set the cursor in XAML, you don't need to use the Cursors class directly.

```
<Button Cursor="Help">Help</Button>
```

Content Controls

- A *content control* is a still more specialized type of control that is able to hold (and display) a piece of content.
- Technically, a content control is a control that can contain a single nested element.
- The Content property supports any type of object, but it separates objects into two groups and gives each group different treatment:
 - ▣ *Objects that don't derive from UIElement*: The content control calls ToString() to get the text for these controls and then displays that text.
 - ▣ *Objects that derive from UIElement*: These objects (which include all the visual elements that are a part of WPF) are displayed inside the content control by using the UIElement.OnRender() method.

Content Controls

- To understand how this works, consider the humble button.

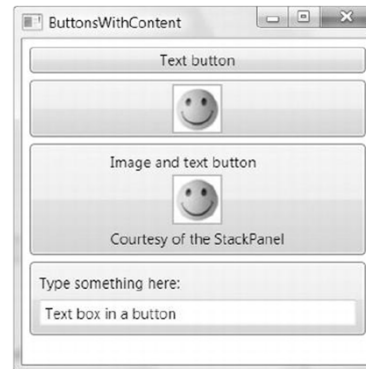

```
<Button Margin="3">Text content</Button>
```
- For example, you can place an image inside a button by using the Image class:


```
<Button Margin="3">
  <Image Source="happyface.jpg" Stretch="None" />
</Button>
```
- Or you could combine text and images by wrapping them all in a layout container such as the StackPanel:

```
<Button Margin="3">
  <StackPanel>
    <TextBlock Margin="3">Image and text button</TextBlock>
    <Image Source="happyface.jpg" Stretch="None" />
    <TextBlock Margin="3">Courtesy of the StackPanel</TextBlock>
  </StackPanel>
</Button>
```

Content Controls

- If you wanted to create a truly exotic button, you could even place other content controls such as text boxes and buttons inside the button.



Other Controls (Text Book)

- The CheckBox
- The RadioButton
- Tooltips
- The Popup
- The TabItem
- The GroupBox
- List Controls
- Date Controls
- The ProgressBar

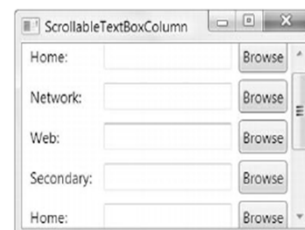
Scroll Viewer

- Scrolling is a key feature if you want to fit large amounts of content in a limited amount of space.
- In order to get scrolling support in WPF, you need to wrap the content you want to scroll inside a ScrollViewer.
- **Example : To make a Grid scrollable**, you simply need to wrap the Grid in a ScrollViewer, as shown in this slightly shortened markup:

Scroll Viewer

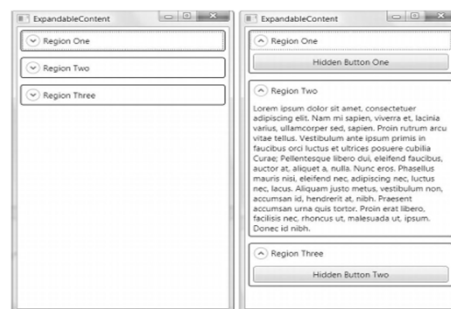
```
<ScrollViewer>
  <Grid Margin="3,3,10,3">
    <Grid.RowDefinitions>
      ...
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      ...
    </Grid.ColumnDefinitions>

    <Label Grid.Row="0" Grid.Column="0" Margin="3"
      VerticalAlignment="Center">Home:</Label>
    <TextBox Grid.Row="0" Grid.Column="1" Margin="3"
      Height="Auto" VerticalAlignment="Center"></TextBox>
    <Button Grid.Row="0" Grid.Column="2" Margin="3" Padding="2">
      Browse</Button>
    ...
  </Grid>
</ScrollViewer>
```



The Expander

- It wraps a region of content that the user can show or hide by clicking a small arrow button.
- This technique is used frequently in online help and on web pages, to allow them to include large amounts of content without overwhelming users with information they don't want to see.



The Expander

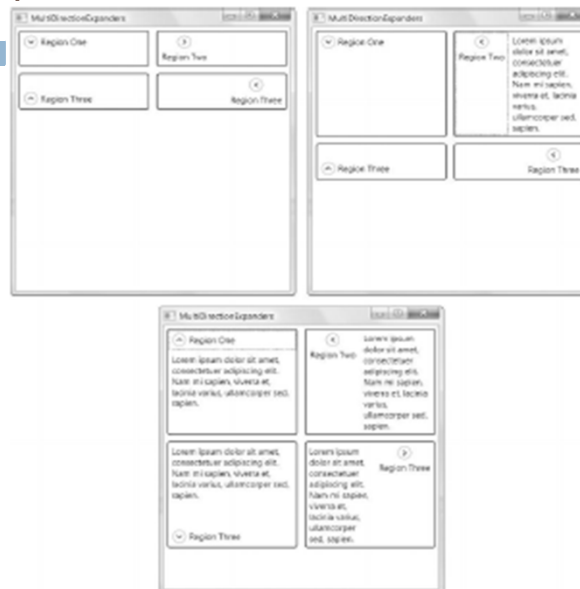
- Using an Expander is extremely simple—you just need to wrap the content you want to make collapsible inside.

```
<StackPanel>
  <Expander Margin="5" Padding="5" Header="Region One">
    <Button Padding="3">Hidden Button One</Button>
  </Expander>
  <Expander Margin="5" Padding="5" Header="Region Two">
    <TextBlock TextWrapping="Wrap">
      Lorem ipsum dolor sit amet, consectetur adipiscing elit ...
    </TextBlock>
  </Expander>
  <Expander Margin="5" Padding="5" Header="Region Three">
    <Button Padding="3">Hidden Button Two</Button>
  </Expander>
</StackPanel>
```

- You can also choose in which direction the expander expands. In previous Figure, the standard value (Down) is used, but you can also set the **ExpandDirection** property to **Up**, **Left**, or **Right**. When the Expander is collapsed, the arrow always points in the direction where it will expand.

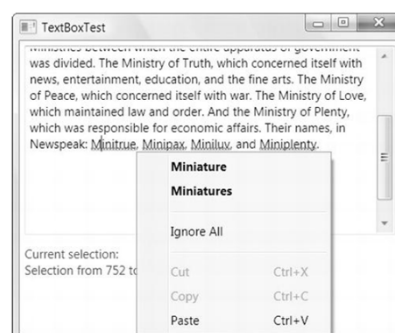
The Expander

□



Spell Checking

- The TextBox includes an unusual frill: an integrated spell-check feature, which underlines unrecognized words with a red squiggly line. The user can right-click an unrecognized word and choose from a list of possibilities, as shown in Figure .



Spell Checking

- To turn on the spell-check functionality for the TextBox control, you simply need to set the SpellCheck.
- **IsEnabled** dependency property, as shown here:

```
<TextBox SpellCheck.IsEnabled="True">...</TextBox>
```

- The spelling checker is WPF-specific and doesn't depend on any other software (such as Office).
- The spelling checker determines which dictionary to use based on the input language that's configured for the keyboard.