WPF Controls

Windows Programming Course

Agenda

1. The Control Class
2. Content Controls
3. Text Controls
4. List Controls
5. Other Controls

1.
The Control Class

WPF Controls

All controls derive from the `System.Windows.Control` class, which adds a bit of basic infrastructure:

- The ability to set the alignment of content inside the control.
- The ability to set the tab order.
- Support for painting a background, foreground, and border.
- Support for formatting the size and font of text content.

Background and Foreground Brushes

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

<Button Background="#FFFF0000">A Button</Button>
```

It's equivalent to this more verbose syntax:

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

Fonts

Name	Description
FontFamily	*The name of the font you want to use*
FontSize	*The size of the font in device-independent units* • **px** (default) is device-independent units (1/96th inch per unit) • **in** is inches; 1in==96px • **cm** is centimeters; 1cm==(96/2.54) px • **pt** is points; 1pt==(96/72) px
FontStyle	*The angling of the text:* Normal | Italic | Oblique
FontWeight	*The heaviness of text:* Thin | ExtraLight | UltraLight | Light | Regular | Bold | …
FontStretch	*The amount that text is stretched or compressed:* Condensed (75.0%) | SemiCondensed (87.5%) | Medium (100.0%) | SemiExpanded (112.5%) | Expanded (125.0%) | ExtraExpanded (150.0%) | UltraExpanded (200.0%)

Fonts – 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. Although the typographic rules and characters for each variation are defined separately, the operating system realizes they are related. As a result, you can configure an element to use Arial Regular, set the FontWeight property to Bold, and be confident that WPF will switch over to the Arial Bold typeface.

E.g.:

```
<Button FontFamily="Times New Roman" FontSize="18">A Button</Button>

<Button FontFamily="Technical Italic, Comic Sans MS, Arial">A Button</Button>
```

Fonts – Text Decorations and Typography

Text Decorations and Typography, e.g.:

```
<TextBlock TextDecorations="Underline">Underlined text</TextBlock>

<TextBlock TextDecorations="Strikethrough">Underlined text</TextBlock>
```

Font Substitution

```
<Button FontFamily="Technical Italic, Comic Sans MS, Arial">A Button</Button>
```

2.
Content Controls

Label
Button
Checkbox
Radio button
Tooltip

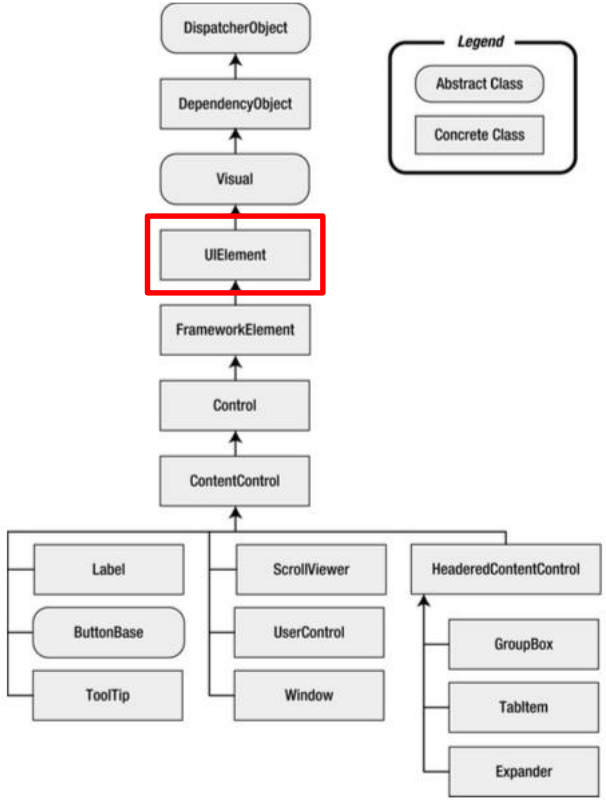
ScrollViewer
GroupBox
TabItem
Expander

Content Controls

A content control is a still more specialized type of control that is able to hold (and display) a piece of content.

The ContentControl class adds a Content property, which accepts a single object. The property supports any type of object:

- Objects that don't derive from UIElement: Call ToString() to get text then displays that text
- Objects that derive from UIElement: Call UIElement.OnRender() method.

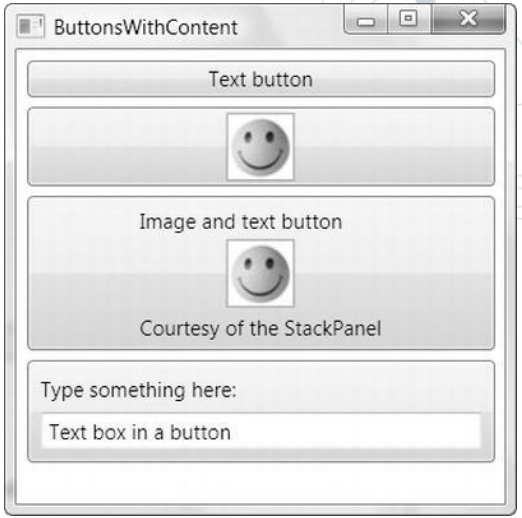


Content Controls – The Content Property

E.g.: WinForms: `Button.Text = "Click me!"`

WPF: `<Button Margin="3" Content="Text content"></Button>`

```
<Button Margin="3">Text content</Button>

<Button Margin="3">
  <Image Source="happyface.jpg" Stretch="None" />
</Button>

<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>
```



The Label

Represents the text label for a control and provides support for access keys.

E.g.: `<Label Content="This is a Label control." />`

The Label control vs. the TextBlock control:

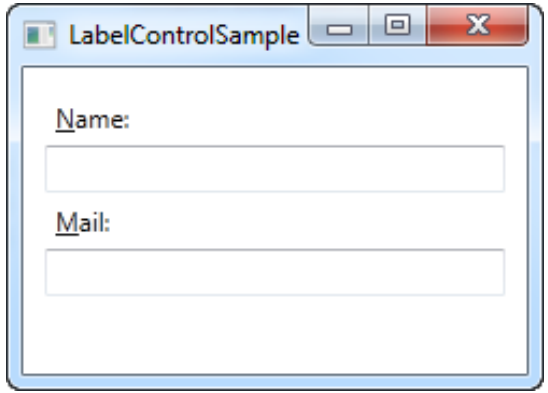
- Specify a border
- Render other controls, e.g. an image
- Use templated content through the ContentTemplate property
- **Use access keys to give focus to related controls.**

The Label

Use access keys to give focus to related controls.

E.g.:

```
<Label Content="_Name:" Target="{Binding ElementName=txtName}" />
<TextBox Name="txtName" />
<Label Content="_Mail:" Target="{Binding ElementName=txtMail}" />
<TextBox Name="txtMail" />
```



The Button

The Button class represents the ever-present Windows push button.

It adds just two writeable properties, IsCancel and IsDefault:

- *When IsCancel is true*, this button is designated as the cancel button for a window (press Esc key).
- *When IsDefault is true*, this button is designated as the default button (also known as the *accept button*, press Enter key).

The ToggleButton and RepeatButton

Along with Button, three more classes derive from ButtonBase. These include the following:

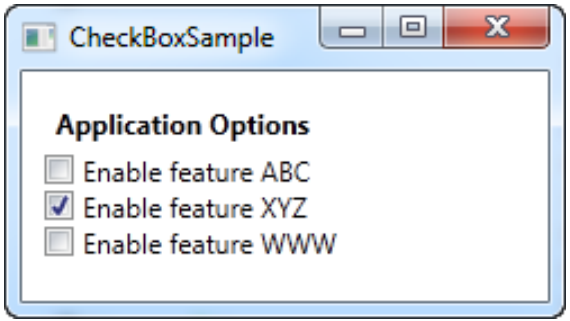
- **RepeatButton** fires Click events continuously, as long as the button is held down. Ordinary buttons fire one Click event per user click.
- **ToggleButton** represents a button that has two states (pushed or unpushed). When you click a ToggleButton, it stays in its pushed state until you click it again to release it. This is sometimes described as *sticky click* behavior.

The CheckBox/RadioButton

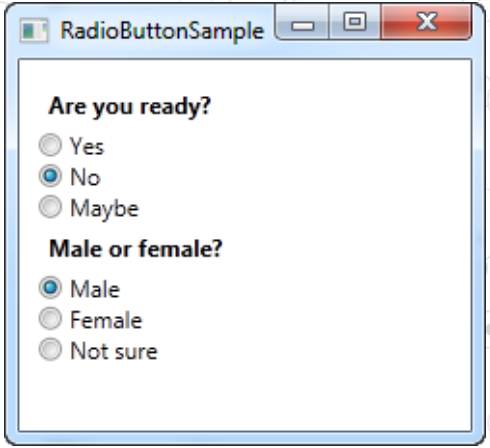
Both the CheckBox and the RadioButton derive from ToggleButton which means they can be switched on or off by the user, hence their “toggle” behavior.

CheckBox example:

```
<Label FontWeight="Bold">Application Options</Label>
<CheckBox>Enable feature ABC</CheckBox>
<CheckBox IsChecked="True">Enable feature XYZ</CheckBox>
<CheckBox IsThreeState="True">Enable feature WWW</CheckBox>
```



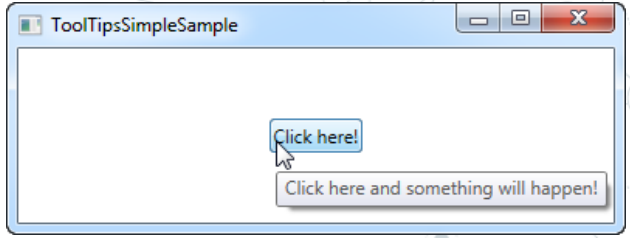
The CheckBox/RadioButton (cont.)



RadioButton example:

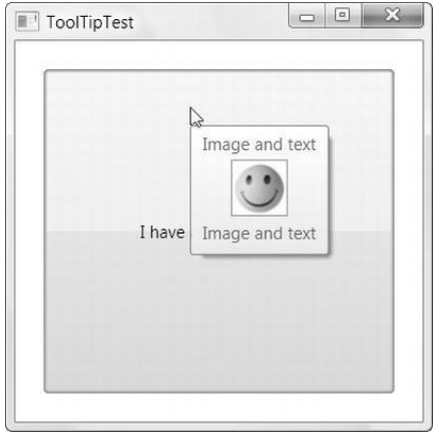
```
<Label FontWeight="Bold">Are you ready?</Label>
<RadioButton GroupName="ready">Yes</RadioButton>
<RadioButton GroupName="ready">No</RadioButton>
<RadioButton GroupName="ready" IsChecked="True">Maybe</RadioButton>

<Label FontWeight="Bold">Male or female?</Label>
<RadioButton GroupName="sex">Male</RadioButton>
<RadioButton GroupName="sex">Female</RadioButton>
<RadioButton GroupName="sex" IsChecked="True">Not sure</RadioButton>
```

Tooltips



E.g.:

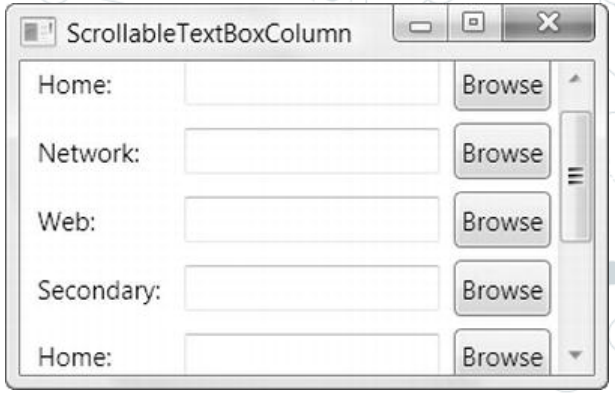
```
<Button ToolTip="Click here and something will happen!">Click
here!</Button>
<Button>
  <Button.ToolTip>
    <StackPanel>
      <TextBlock Margin="3" >Image and text</TextBlock>
      <Image Source="happyface.jpg" Stretch="None" />
      <TextBlock Margin="3" >Image and text</TextBlock>
    </StackPanel>
  </Button.ToolTip>
  <Button.Content>I have a fancy tooltip</Button.Content>
</Button>
```



The ScrollViewer

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.

The ScrollViewer (cont.)



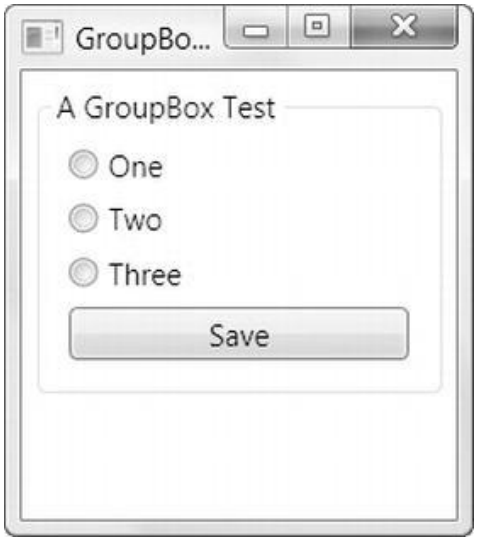

```
<ScrollViewer>
  <Grid Margin="3,3,10,3">
  <Grid.RowDefinitions> ... </Grid.RowDefinitions>
  <Grid.ColumnDefinitions> ... </Grid.ColumnDefinitions>
  <Label Grid.Row="0" Grid.Column="0" Margin="3”
         VerticalAlignment="Center">Home:</Label>
  <TextBox Grid.Row="0" Grid.Column="1" Margin="3" Height="Auto"
         VerticalAlignment="Center"></TextBox>
  <Button Grid.Row="0" Grid.Column="2" Margin="3" Padding="2"> Browse</Button>
  ...
  </Grid>
</ScrollViewer>
```

The GroupBox

The GroupBox control derives from HeaderedContentControl.

E.g.:

```
<GroupBox Header="A GroupBox Test" Padding="5" Margin="5”
VerticalAlignment="Top">
<StackPanel>
  <RadioButton Margin="3">One</RadioButton>
  <RadioButton Margin="3">Two</RadioButton>
  <RadioButton Margin="3">Three</RadioButton>
  <Button Margin="3">Save</Button>
</StackPanel>
</GroupBox>
```

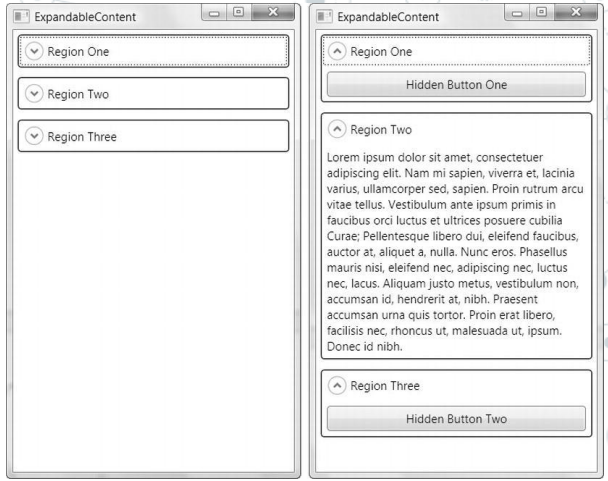


The TabItem

E.g.:

```
<TabControl Margin="5">
  <TabItem Header="Tab One">
    <StackPanel Margin="3">
      <CheckBox Margin="3">Setting One</CheckBox>
      <CheckBox Margin="3">Setting Two</CheckBox>
      <CheckBox Margin="3">Setting Three</CheckBox>
    </StackPanel>
  </TabItem>
  <TabItem Header="Tab Two">...</TabItem>
</TabControl>
```

The Expander



E.g.:

```
<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, consectetuer adipiscing elit ...
    </TextBlock>
  </Expander>
  <Expander Margin="5" Padding="5" Header="Region Three">
    <Button Padding="3">Hidden Button Two</Button>
  </Expander>
</StackPanel>
```

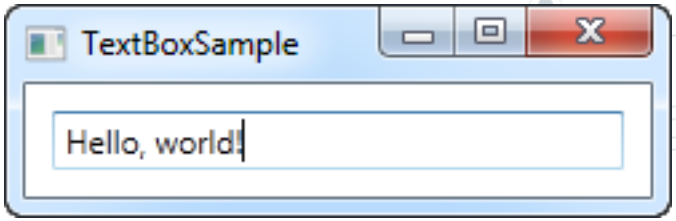
2.
Text Controls

TextBox
RichTextBox
PasswordBox

The TextBox

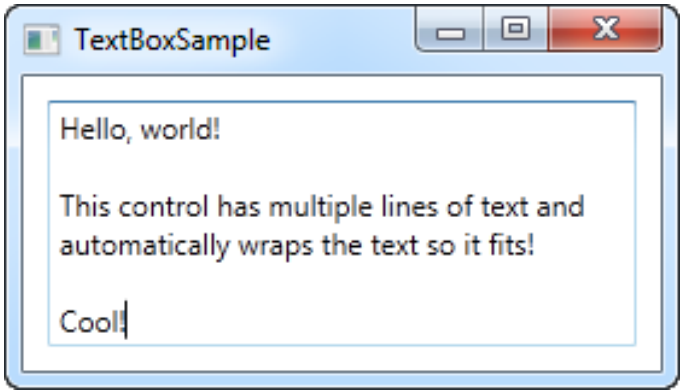
Single-line TextBox example:

```
<TextBox Text="Hello, world!" />
```



Multi-line TextBox example:

```
<TextBox AcceptsReturn="True" TextWrapping="Wrap" />
```



The RichTextBox

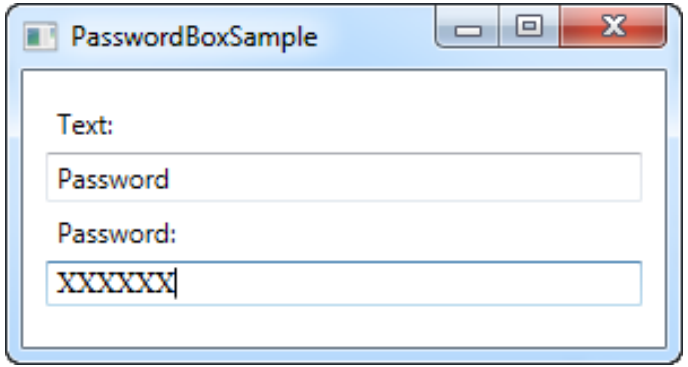
E.g.:

```
<RichTextBox Margin="10">
  <FlowDocument>
    <Paragraph FontSize="36">Hello, world!</Paragraph>
    <Paragraph FontStyle="Italic" TextAlignment="Left"
FontSize="14" Foreground="Gray">Thanks to the RichTextBox
control, this FlowDocument is completely editable!
    </Paragraph>
  </FlowDocument>
</RichTextBox>
```

The PasswordBox

E.g.:

```
<Label>Text:</Label>
<TextBox />
<Label>Password:</Label>
<PasswordBox MaxLength="6" PasswordChar="X" Password="123456" />
```



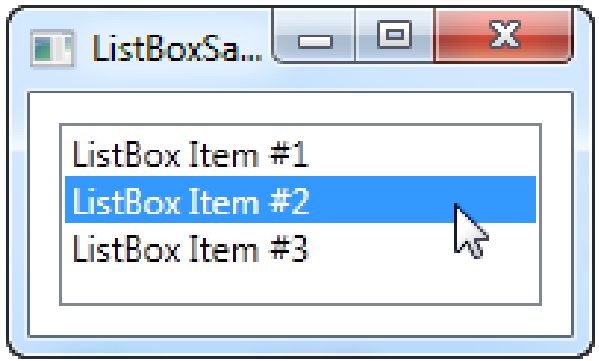
3.
List Controls

List box
ComboBox

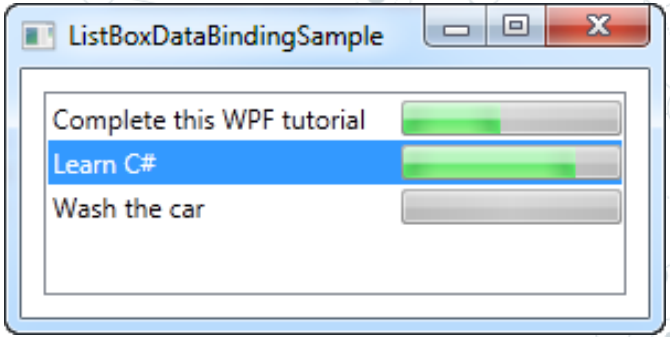
The ListBox

List Controls derive from the ItemsControl class (which itself derives from Control). E.g.:

```
<ListBox>
  <ListBoxItem>ListBox Item #1</ListBoxItem>
  <ListBoxItem>ListBox Item #2</ListBoxItem>
  <ListBoxItem>ListBox Item #3</ListBoxItem>
</ListBox>
```



The ListBox (cont.)

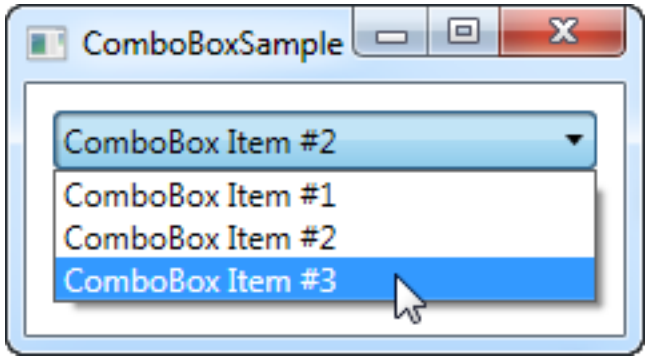



```
<ListBox HorizontalContentAlignment="Stretch”
         ItemsSource="{Binding data}" >
  <ListBox.ItemTemplate>
    <DataTemplate>
      <Grid Margin="0,2">
        <Grid.ColumnDefinitions>...</Grid.ColumnDefinitions>
        <TextBlock Text="{Binding Title}" />
        <ProgressBar Grid.Column="1" Minimum="0" Maximum="100" Value="{Binding
Completion}" />
      </Grid>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

The ComboBox

The ComboBox control is in many ways like the ListBox control, but takes up a lot less space, because the list of items is hidden when not needed. E.g.:

```
<ComboBox>
   <ComboBoxItem>ComboBox Item #1</ComboBoxItem>
   <ComboBoxItem IsSelected="True">ComboBox Item #2</ComboBoxItem>
   <ComboBoxItem>ComboBox Item #3</ComboBoxItem>
</ComboBox>
```

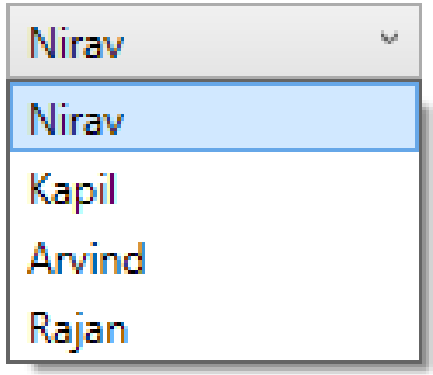


The ComboBox (cont.)

Data binding & Selected Item

```
<ComboBox ItemsSource="{Binding Path=Persons}"
          SelectedItem="{Binding Path=SPerson}"
          DisplayMemberPath="Name"/>
```

```
public class Person {
  public string Name {get; set;}
  public int Id {get; set;}
}
```



4.
Other Controls

Windows
Menu/ContextMenu

The Window Control

The Window class derives from ContentControl. That means it can contain a single child (usually a layout container such as the Grid control).

Name	Description
Title	The caption that appears in the title bar for the window (and in the taskbar).
Icon	Icons appear at the top left of a window (if it has one of the standard border styles), in the taskbar (if ShowInTaskBar is true).
ResizeMode	CanResize | CanResizeWithGrip | NoResize | CanMinimize
ShowInTaskbar	If set to true, the window appears in the taskbar and the Alt+Tab list.
SizeToContent	Allows to create a window that enlarges itself automatically.
WindowStartupLocation	CenterOwner | CenterScreen | Manual
WindowState	Normal | Minimized | Maximized

The Window Control (cont.) – Showing a Window

Showing a *modal* window. Modal windows stop the user from accessing the parent window by blocking any mouse or keyboard input to it, until the modal window is closed.

```
TaskWindow winTask = new TaskWindow();
winTask.ShowDialog();
```

Showing a *modeless* window:

```
MainWindow winMain = new MainWindow();
winMain.Show();
```

The Window Control (cont.) –Window Ownership

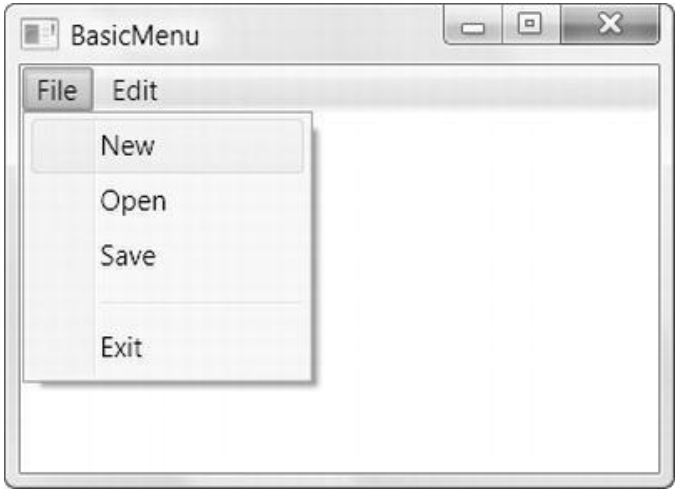
Owned windows are useful for floating toolbox and command windows. When an owner window is minimized, the owned windows are also minimized automatically. When an owned window overlaps its owner, it is always displayed on top.

```
// Create a new window.
ToolWindow winTool = new ToolWindow();
// Designate the current window as the owner.
winTool.Owner = this;
// Show the owned window.
winTool.Show();
```

The Menu Control

Menus are composed of MenuItem objects and Separator objects. The MenuItem class derives from HeaderedItemsControl.

```
<Menu>
  <MenuItem Header="File">
    <MenuItem Header="New"></MenuItem>
    <MenuItem Header="Open"></MenuItem>
    <MenuItem Header="Save"></MenuItem>
    <Separator></Separator>
    <MenuItem Header="Exit"></MenuItem>
  </MenuItem>
  <MenuItem Header="Edit">...
  </MenuItem>
</Menu>
```



The ContextMenu Control

Like the Menu, the ContextMenu class holds a collection of MenuItem objects. The difference is that a ContextMenu can't be placed in a window. Instead, it can be used only to set the ContextMenu property of another element:

```
<TextBox>
  <TextBox.ContextMenu>
    <MenuItem ... >
      ...
    </MenuItem>
  </TextBox.ContextMenu>
</TextBox>
```

Thanks!

Any questions?

You can find me at:

tranminhphuoc@gmail.com