Windows Presentation Foundation (WPF)

Introduction to WPF

- This Section covers:
  - What is WPF?
  - WPF Window
  - Application.XAML
  - Controls
  - Resource Example
  - Routed Events

What is WPF?

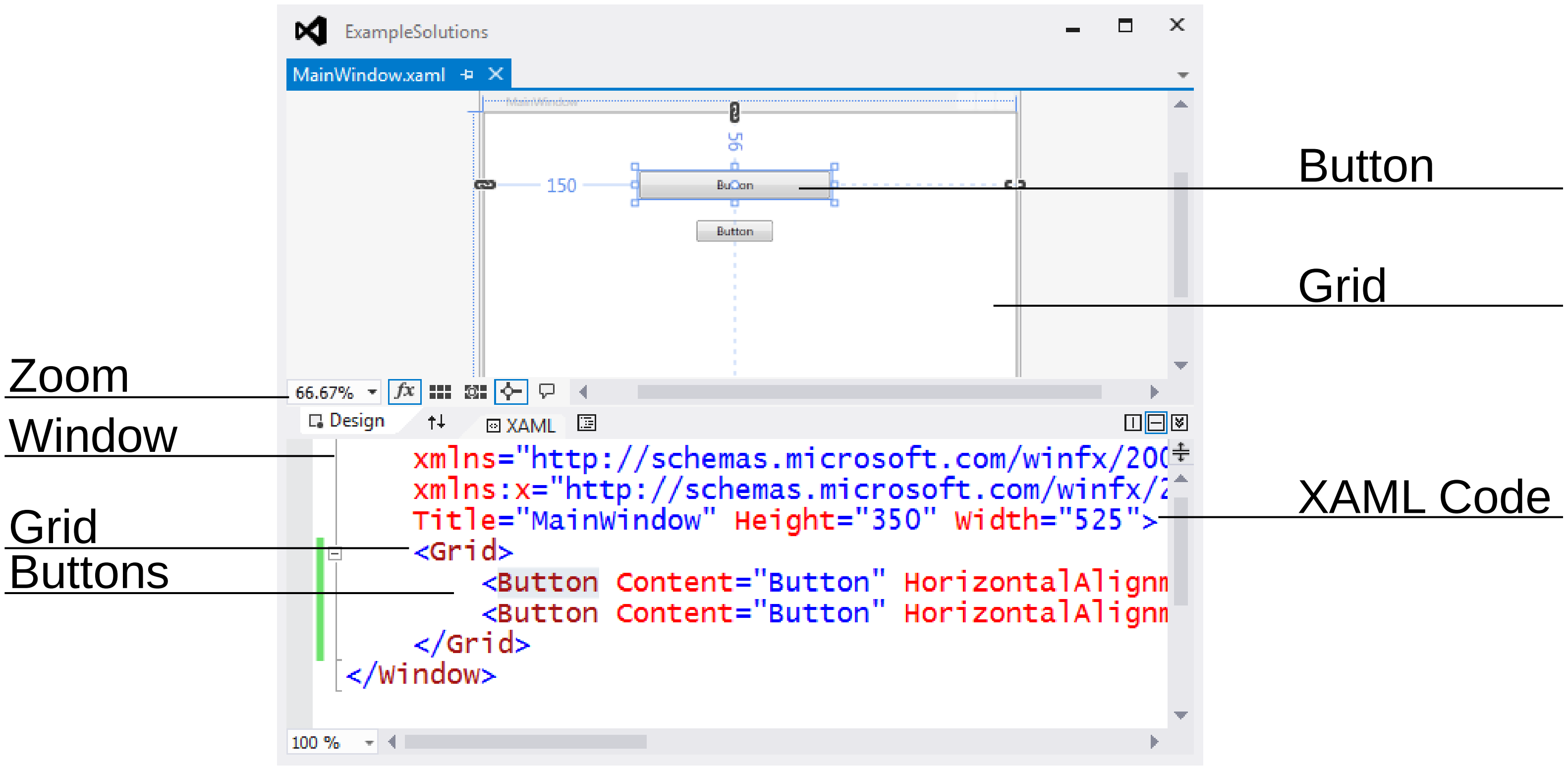
- Windows Presentation Foundation allows creation of Rich GUI Interfaces
- WPF provides an alternative class library to WinForms
  - Not bound to traditional Win32 GUI Presentation
  - More flexible in design of layout
  - More flexible in binding to data
  - More options in styling and colour schemes
- Allows both:
  - Imperative programming style (traditional coding)
  - Declarative programming style (XAML)

WPF Application

- WPF Applications can be created in a similar way to WinForms Application, but using a different Designer
- Visual Studio .NET provides a RAD style environment for WPF Applications
  - Drag and Drop can be used (to some extent!!)
  - Double clicking on element will add default event handler
- WPF also allows use of Model View ViewModel style application
  - Bind Commands instead of adding event handlers
  - More flexible and aides testing
- WinForms Designer generated code was placed in the Form's class
- WPF Designer generated code uses XAML (XML) to define the user interface
  - XAML is the means of expressing the design
  - Much of this could also be done using code

WPF Window

- Designing a Window in Visual Studio.NET



```
    xmlns="http://schemas.microsoft.com/winfx/200
    xmlns:x="http://schemas.microsoft.com/winfx/2
    Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button Content="Button" HorizontalAlignm
        <Button Content="Button" HorizontalAlignm
    </Grid>
</Window>
```

Application.XAML

- WPF Applications have XAML file for the Application itself:

```
<Application x:Class="Application"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml"
    >
    <Application.Resources>

    </Application.Resources>
</Application>
```

Name for Resulting Class, i.e. Name of Code Behind Class

XAML Namespace

WPF Namespace

XAML File for Startup Window

- Need appropriate Namespaces

Many Familiar Looking ‘Controls’

- Many UIElements appear familiar:
  - Label, Button, TextBox, etc.
- Label and Button are ContentControls
  - Have Content property
  - Can be associated with Text or an object
  - Allows hierarchy (nesting) of content
    - Button within Button!!!
    - Image and text within Button!
- UIElement use Dependency Properties
  - More capable that standard properties

Resource Example

- The Background Property set from a Resource:

Define Resource For Use In Grid

Key Needed for Identification

Top Left Hand Corner

Bottom Right Hand Corner

0 to 1 For Relative Position of Colour

Select Resource for Background

```
<Grid>
  <Grid.Resources>
   <LinearGradientBrush x:Key="SalmonYellow" StartPoint="0 0" EndPoint="1 1">
    <GradientStop Color="Salmon" Offset="0"/>
    <GradientStop Color="Yellow" Offset="1"/>
   </LinearGradientBrush>
  </Grid.Resources>

  <Button Height="23" Background="{StaticResource SalmonYellow}"
              Name="Button1" VerticalAlignment="Center" HorizontalAlignment="Center"
              Click="Button1_Click" >Button
  </Button>
</Grid>
```

Routed Event Handling

- Event handlers may be placed at a number of levels within hierarchy:

```
<StackPanel Name="stackPanel1" Orientation="Vertical"
            ButtonBase.Click="CommonButton_Click">
```

Event Handler Common to Buttons

```
  <Button Height="32" Name="button1" Width="124" >
    <Button Height="22" Width="80" Content="Greetings"
            Click="Button_Click"/>
  </Button>
      ...
```

Button Click Event Handler

Both Event Handlers May Be Executed

Handling Events

- Event Handlers have second parameter inheriting from RoutedEventArgs:

```
private void Button1_Click( System.Object sender,
                                          System.Windows.RoutedEventArgs e)
{
  MessageBox.Show("You clicked?");

  e.Handled = true;
}
```

Consider This Event Handled

Introduction to WPF - Summary

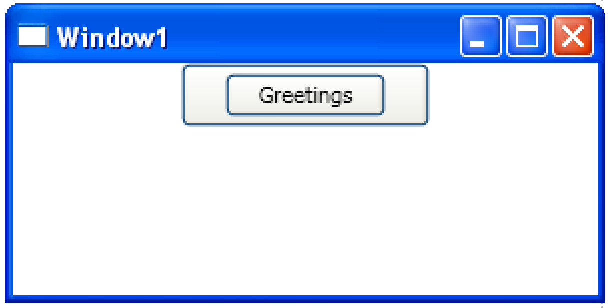
- This Section covered:
  - What is WPF?
  - WPF Window
  - Application.XAML
  - Controls
  - Resource Example
  - Routed Events

Controls, Appearance and Commands

- The Section Covers:
  - WPF Button
  - Grid
  - StackPanel
  - Data Binding
  - Validation
  - Menu
  - Commands

WPF Button

- The following code illustrates nesting one Button within another (why?):



```
Button b = new Button();
b.Width = 80;
b.Height = 22;
b.Content = "Greetings“;

button1.Content = b;
```

- Or using XAML

```
<Button Height="32" Name="button1" Width="124" >
          <Button Height="22" Width="80" Content="Greetings"/>
</Button>
```

Windows and Dialog

- Windows can be displayed Modally or Non-modally:
  - ShowDialog() or Show()
- ShowDialog() method returns a nullable bool
  - Use DialogResult property of Window to set true (OK) or false (Cancel)

```
<Button Content="Cancel" Height="23" Margin="203,184,0,0"
  Name="buttonCancel" Width="75" IsCancel="True" />
```

Cancel Button Sets DialogResult to false

```
<Button Content="OK" Height="23" Margin="202,225,0,0"
  Name="buttonOK"  Width="75" IsDefault="True"
  IsCancel="False" />
```

Default Button

Displaying Windows and Dialogs

- Applications typically display many Windows
- To display a window, instantiate the window and then:
  - Show() method – displays as Modeless Window
  - ShowDialog() method – display as a Modal dialog
    - Return value indicates success or failure!

```
AWindow aw = new AWindow();

if( aw.ShowDialog() ?? false)
{
      ...
}
```

WPF Panel Types

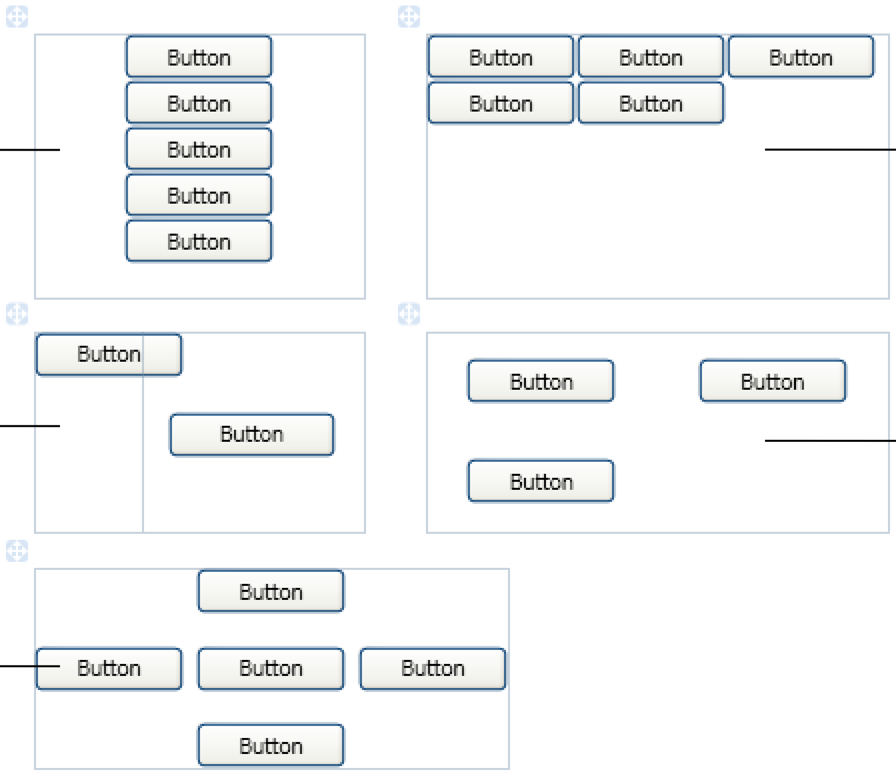
StackPanel

WrapPanel

Grid

UniformGrid

DockPanel



Grid Control

- The Grid Control allows display of data within a Grid
- Within XAML the rows and columns are defined as below:

```
<Grid ShowGridLines="False">
        <Grid.RowDefinitions>
            <RowDefinition />                 Two Rows
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />              Two Columns
        </Grid.ColumnDefinitions>
       …
```

- Child controls can be associate with individual grid positions by setting the Row and Column properties
- Controls may span rows or columns using the RowSpan or ColumnSpan properties

StackPanel

- The StackPanel displays a collection of UIElements
  - Can be added using imperative style:

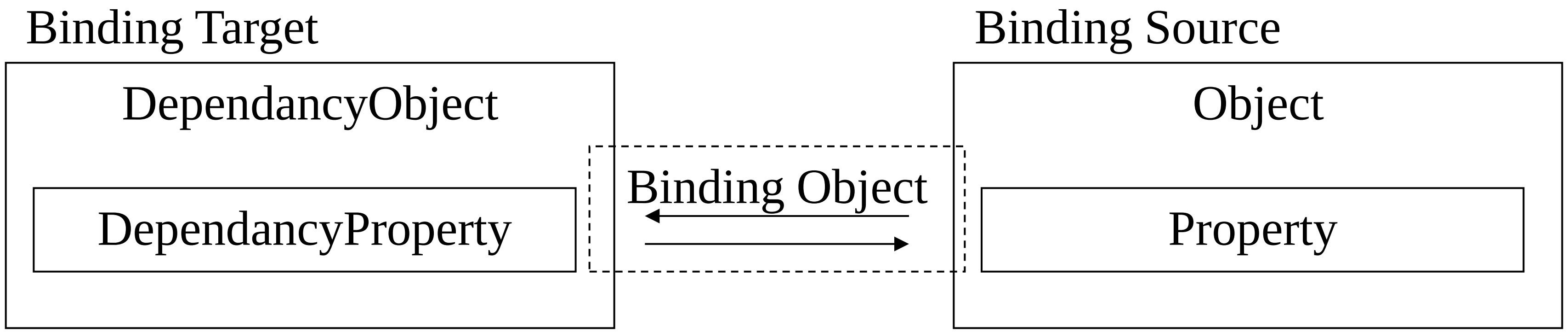
```
stackPanel2.Children.Add(new TextBox()
            { Height = 22, Width = 80, Text = "Hello" });
tackPanel2.Children.Add(new Label()
            { Height = 22, Width = 80, Content = "Hello" });
```

  - Or declarative style

```
<StackPanel Height="100" Name="stackPanel2" Width="200" >
          <TextBox Height="22" Width="80" Text="Hello" />
          <Label  Height="22" Width="80" Content="Hello" />
</StackPanel>
```

DataBinding Principles

- Data binding is relatively flexible in WPF
  – Data may be propagated:
    - One way from the ‘source’
    - One way to the ‘source’
    - Two way



Binding Sources

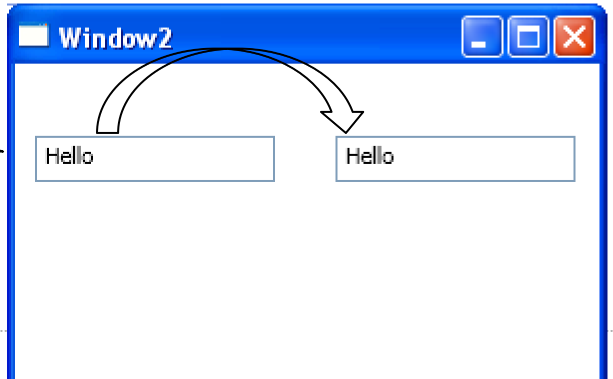
- A wide range of data types can be bound to WPF controls
  - Simple CLR types; DataSets/DataTables; Collections and XML
- Either:
  - Associate object with DataContext
    - Bind ‘Path’ to property
  - Associate collection with ItemsSource
    - Define DataTemplate and Bind ‘Path’ to property
- Triggering determined by using UpdateSourceTrigger property:
  - PropertyChanged
  - LostFocus
  - Explicit
- Most DependencyProperties have the default of PropertyChanged

Binding Between Elements

- Binding can also be added between elements on a Window, as follows:

```
<TextBox Height="23" HorizontalAlignment="Left"
        Margin="10,36,0,0" Name="textBox1"
        VerticalAlignment="Top" Width="120" />
<TextBox Height="23" HorizontalAlignment="Right"
        Margin="0,36,12,0" Name="textBox2"
        VerticalAlignment="Top" Width="120"
        Text="{Binding ElementName=textBox1,Path=Text}"/>
```

Property Change Causes Update



Binding to a Collection

- When binding collections the ItemsSource property is used:

```
class Window1
{
  ObserverableCollection<SomeData> data =
                new ObserverableCollection<SomeData>();
  public Windows1()
  {
    // This call required by Windows Form Designer.
    InitializeComponent();

    for( int i = 0; i < 10; ++i)
    {
      data.Add(new SomeData("Fred" + i.ToString(), i));
    }

    listBox1.ItemsSource = data;
```

Collection for Data

Populate Collection with Data

Collection bound to ListBox

Displaying ListBox Items

- Define ListBox ItemTemplate for displaying items:

```
<ListBox Margin="44,97,50,65" Name="listBox1" Grid.ColumnSpan="3">
       <ListBox.ItemTemplate>
           <DataTemplate>

               <StackPanel Orientation="Horizontal" Margin="5">
                   <TextBox Text="{Binding Path=Name}" Margin="5,0,5,0" />
                   <TextBox Text="{Binding Path=Val}" />
               </StackPanel>
           </DataTemplate>
       </ListBox.ItemTemplate>
</ListBox>
```

Display Individual element within StackPanel

Binding Path to Properties on Source Objects

Validation

- Silverlight provides a number of mechanisms for validation:
  - Throw an exception if bound property ‘fails’ validation!
    - ValidatesOnException
  - Implement IDataErrorInfo
    - ValidatesOnDataErrors
  - Implement INotifyDataErrorInfo (.NET 4.5)
    - ValidatesOnNotifyDataErrors

Validating On Exception

- Property throwing exception:

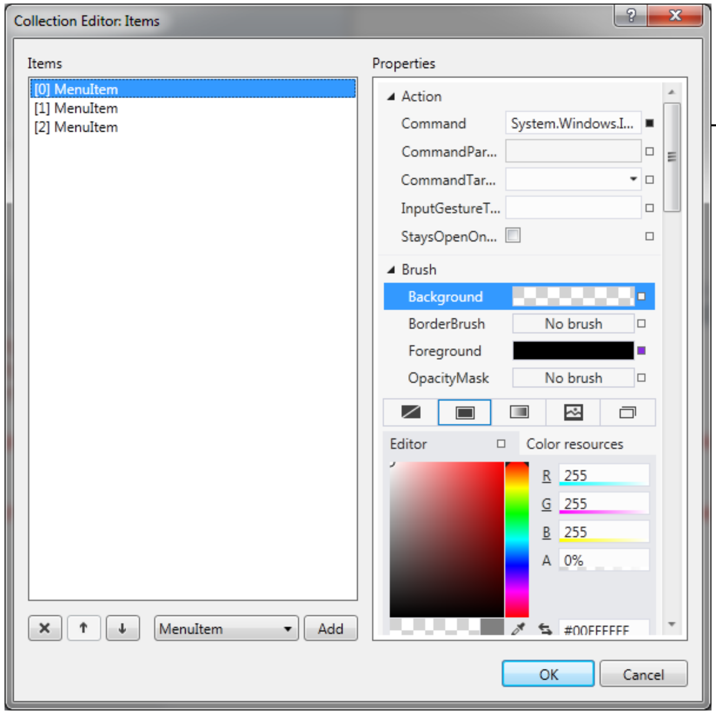
```
public class SomeData
{
  private string _name = string.Empty;
  private int _val = 0;
  public int Val
  {
    get { return _val; }
    set
    {
      if (value < 0) throw new Exception("Number too small!");
      _val = value;
    }
  }
}
```

Menu

- Add Menu Element to Panel:

```
<Menu  Height="22" Name="Menu1" VerticalAlignment="Top">
            …
</Menu>
```

- Menu Items/Context Menu Items can be edited using the Collections Editor:



Collection Editor: Items

Items

[0] MenuItem
[1] MenuItem
[2] MenuItem

Properties

Action

Command	System.Windows.I...
CommandPar...	
CommandTar...	
InputGestureT...	
StaysOpenOn...	☐

Brush

Background	▨
BorderBrush	No brush
Foreground	■
OpacityMask	No brush

Editor | Color resources

R 255
G 255
B 255
A 0%
#00FFFFFF

Predefined Command

Menus and Commands

- There are many standard Commands defined:
  - E.g. Cut, Copy, Paste (ApplicationCommands)
  - MenuItems and Buttons can be a command Source
  - These can be associated with Menus:

```
<Menu  Height="22" Name="Menu1" VerticalAlignment="Top">
        <MenuItem Header="_Edit">
                <MenuItem Command="ApplicationCommands.Cut" />
                <MenuItem Command="ApplicationCommands.Copy" />
                <MenuItem Command="ApplicationCommands.Paste" />
        </MenuItem>
</Menu>
```

Predefined Commands

- Without setting a CommandTarget the target is the element with keyboard focus

Defining A Command

- Create a Class implementing ICommand:

```
public class CustomCommand : ICommand
{
  public CustomCommand()
  {
  }
  public bool CanExecute(object parameter)
  {
    return true;
  }
  public event EventHandler CanExecuteChanged;
  public void Execute(object parameter)
  {
    // Do whatever is necessary
  }
}
```

Define Delegate Command!

- Useful class:

Methods can be defined within ViewModel and used to initialize DelegateCommand object

```
class DelegateCommand: ICommand
{
  private Action<object> _act;
  private Predicate<object> _pred;
  public DelegateCommand(Action<Object> act,
                         Predicate<Object> pred)
  {
    _act = act;
    _pred = pred;
  }
  public bool CanExecute(object parameter)
  {
    return _pred(parameter);
  }
  public event EventHandler CanExecuteChanged;
  public void Execute(object parameter)
  {
    _act(parameter);
  }
}
```

Controls Appearance and Commands - Summary

- The Section Covered:
  - WPF Button
  - Grid
  - StackPanel
  - Data Binding
  - Validation
  - Menu
  - Commands