Object Oriented Software Development

9. Creating Graphical User Interfaces

User interfaces

- There are many ways in which people can interact with software systems, including:
  - "Windows" desktop interfaces (GUI)
  - Web interfaces (browser-based)
  - Text interfaces (command-line)
  - Phone interfaces
  - Device interfaces (e.g. washing machine, set-top box)
- Some software is written as a component in a system and has no direct interface with users

Graphical User Interfaces (GUIs)

- Allow users to interact with system through graphical elements - icons, buttons, drop-down menus etc.
- Typically desktop applications which run on windowing environments – Windows, Mac, Linux (KDE/Gnome)
- GUI toolkits allow developers to create GUI applications using pre-defined components, or **controls**

.NET GUI Toolkits

- Windows Forms
  - long-established UI technology
  - graphical application programming interface
  - UI defined in programming language code or using Visual Studio designer
- Windows Presentation Foundation (WPF)
  - Microsoft's newer UI technology for desktop applications
  - UI defined using markup language (XAML) or using Visual Studio designer

.NET GUI Toolkits

- Windows Store Applications
  - UI technology for apps which run on Windows 8 "modern" UI
  - Optimised for touch-based interfaces
  - UI defined using either of a choice of markup languages
    - XAML – used with C#, like WPF
    - HTML – used with JavaScript
  - WPF still used for Windows 8 desktop applications

WPF rationale

- UI layout and design separated from functionality
- Markup language (XAML) for design, programming language (C#, VB, etc) for functionality
- Designers and developers can use separate specialised tools to work on the same project:
- Expression Blend for designers
- Visual Studio for developers

XAML

- A specific type of XML
- Elements defined by tags inside <brackets>
- Elements can have attributes
  - <Button Name="cmdAnswer“>
- Elements can be nested inside other elements
- Elements must have closing tags
  - </Button>

WPF Rationale

- Similar technology, based on XAML and C#/VB, can be used for different interface types:
  - Windows (WPF)
  - Web (Silverlight)
  - Phone (Silverlight)
- Basic idea of building interface using markup and code is similar to some other web development technologies, e.g. HTML & JavaScript, ASP.NET & C#

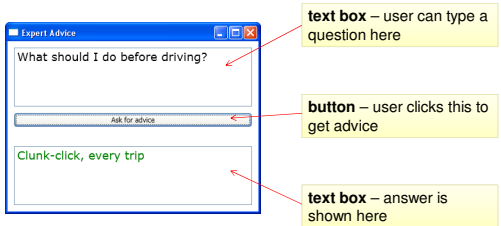
What do GUI toolkits do?

- Provide **controls** with specific appearance and capabilities
  - e.g. A button control looks like this  and can be clicked
  - clicking is a **event** associated with the button
- Provide a way of responding to user actions
  - Can write code which runs when button is clicked
  - Code is attached to button as **event handler**
- Render controls and fire events in response to user actions

Types of control

- Layout controls
  - containers for other controls to position them in the interface
- Interactive controls
  - buttons, combo boxes, check boxes, etc.
- Display controls
  - text, drawing, data
- Application controls
  - menus, toolbars

A simple WPF example



text box – user can type a question here

button – user clicks this to get advice

text box – answer is shown here

XAML file

- Window defined in a XAML file (*Advice.xaml*)
- **Grid** control as a container

```
<Window x:Class="Advice.Advisor"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Expert Advice" Height="300" Width="400" >
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    other controls are defined here, inside <Grid> control

  </Grid>
</Window>
```

grid has 3 rows, middle row sized to fit contents, others expand to fill available space

XAML controls

- Controls defined inside <Grid> element

name of method to handle **click event**

```
<TextBox Margin="10,10,10,10" Name="txtQuestion"
         TextWrapping="Wrap" FontFamily="Verdana" FontSize="18"
         Grid.Row="0" >
  [What's your problem?]
</TextBox>
<Button  Margin="10,0,10,20" Name="cmdAnswer"
         Click="cmdAnswer_Click"
          Grid.Row="1">
  Ask for advice
</Button>
<TextBox Margin="10,10,10,10" Name="txtAnswer"
         TextWrapping="Wrap" FontFamily="Verdana" FontSize="18"
         Foreground="Green"
         Grid.Row="2">
</TextBox>
```

Grid.Row attribute specifies which row of grid control is displayed in

Attributes control appearance of controls (fonts, margins, etc)

Code-behind file

- Contains a C# class which is derived from **Window** library class

```
public partial class Advisor : Window
{
    public Advisor()
    {
        InitializeComponent();
    }

    private void cmdAnswer_Click(object sender, RoutedEventArgs e)
    {
        AdviceGenerator generator = new AdviceGenerator();
        txtAnswer.Text = generator.GetRandomAnswer(txtQuestion.Text);
    }
}
```

constructor

event handler method

event handler method uses a **model class** *AdviceGenerator* and sets **Text property** of the text box named *txtAnswer*

Model class

- Ordinary C# class

```
class AdviceGenerator
{
    string[] answers = new string[]{
        "Dial 999 and ask for the Coastguard",
        "Learn to swim, young man",
        "Clunk-click, every trip",
        "Always remember the Green Cross Code",
        "Think once, think twice, think bike",
        "Switch off some power - now",
        "Be smart..be safe"
        };

    public string GetRandomAnswer(string question)
    {
        Random rnd = new Random();

        return answers[rnd.Next(0, answers.Length)];
    }
}
```

called by event handler

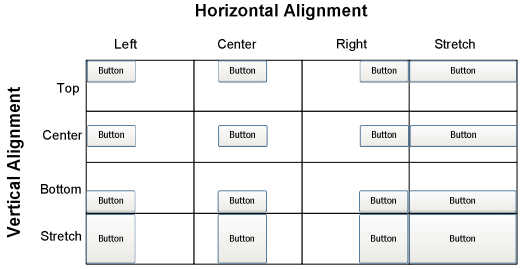
Code and visual designers

- WPF windows can be designed using visual design tools in Visual Studio and Expression Blend
- Important to understand XAML code to get fine control over design
- Plan out layout using capabilities of layout controls rather than dragging controls from designer toolbox and positioning visually

Layout controls

- Grid
  - arranges its child controls in a tabular structure
- Stack Panel, Wrap Panel
  - stacks child elements below or beside each other, Wrap Panel wraps to new line if no space
- Dock Panel
  - docks elements to left, right, top, bottom or centre
- Canvas
  - Elements positioned by coordinates, mainly used for 2D drawing

Alignment



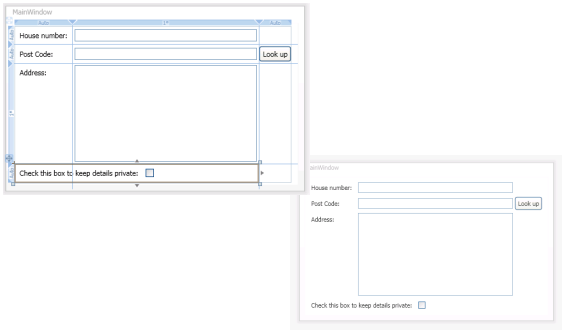
Margin and padding

- The Margin is the extra space **around** the control
- The Padding is extra space **inside** the control
- The Padding of an outer control is the Margin of an inner control

Laying out a grid

- Row and column definitions
- Sizes:
  - **Fixed**: Fixed size
  - **Auto:** Takes as much space as needed by the contained control
  - **Star (*):** Takes as much space as available
- Position each control in grid with properties **Grid.Column** and **Grid.Row**
- Merge grid cells with **Grid.ColumnSpan** and **Grid.RowSpan**

Layout example



Layout example - Grid

4 rows, 3 columns

```
<Grid Margin="3,3,10,3">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
        <RowDefinition Height="Auto"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"></ColumnDefinition>
        <ColumnDefinition Width="*"></ColumnDefinition>
        <ColumnDefinition Width="Auto"></ColumnDefinition>
    </Grid.ColumnDefinitions>
```

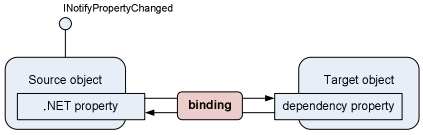

Layout example - controls

can miss out Column="0"

```
    <Label Grid.Row="0" Grid.Column="0" Margin="3"
          VerticalAlignment="Center">House number:</Label>
    <TextBox Grid.Row="0" Grid.Column="1" Margin="3"
          Height="Auto"  VerticalAlignment="Center"></TextBox>
    <Label Grid.Row="1" Margin="3"
          VerticalAlignment="Center">Post Code:</Label>
    <TextBox Grid.Row="1" Grid.Column="1" Margin="3"
          Height="Auto"  VerticalAlignment="Center"></TextBox>
    <Button Grid.Row="1" Grid.Column="2" Padding="2"
            VerticalAlignment="Center">Look up</Button>
    <Label Grid.Row="2" Margin="3"
          VerticalAlignment="Top">Address:</Label>
    <TextBox Grid.Row="2" Grid.Column="1" Margin="3"
          Height="Auto"  VerticalAlignment="Stretch"></TextBox>
    <StackPanel Orientation="Horizontal" Grid.Row="3" Grid.Column="0"
                Grid.ColumnSpan="2">
        <Label Margin="3" VerticalAlignment="Center">
            Check this box to keep details private:</Label>
        <CheckBox Margin="3"
              Height="Auto"  VerticalAlignment="Center"></CheckBox>
    </StackPanel>
</Grid>
```


Binding

- Properties of controls can be automatically updated by properties of other controls or model objects
- Updates can be one-way or two way



Binding controls

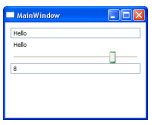
```
<StackPanel Margin="10,10,10,10">
    <TextBox x:Name="txtInput" />
    <Label Content="{Binding ElementName=txtInput,
        Path=Text}" />

    <Slider x:Name="sliderSize" Value="5" />
    <TextBox x:Name="txtSize" Text="{Binding ElementName=sliderSize,
        Path=Value,
        Mode=TwoWay,
        UpdateSourceTrigger=PropertyChanged}" />
</StackPanel>
```

Content property of Label (target) bound to **Text** property of TextBox (source)

Text property of TextBox (target) bound to **Value** property of Slider (source)

Binding mode – changes cause updates both ways



Binding modes

- One time
  - Source property updates target property once and only once
- One way
  - Source property always updates target property
- Two way
  - Source and target properties update each other – change one and the other changes
- One way to source
  - Target property always updates source property

Binding to an object

- Model class – simple Employee class

```
public class Employee : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void Changed(string propertyName)...

    private string name;

    public string Name
    {
        get { return name; }
        set {
            if (name != value)
            {
                name = value;
                Changed("Name:" + name);
            }
        }
    }
}
```

extra code to notify changes in property values for binding (details not shown)

Binding to an object

- XAML – TextBox is bound to Name property

```
<TextBox x:Name="txtName"  Text="{Binding Path=Name,
    Mode=TwoWay}" />
```

- Don't specify source here – it will be the **data context** of the window
- Code-behind – create model object and set it as data context for window


```
public MainWindow()
{
    InitializeComponent();

    Employee emp = new Employee { Name = "Michael" };
    DataContext = emp;
}
```


What's next?

- You will learn how to work with data which is stored on disk - in files and in databases - and use WPF data binding to display data in an application