module 14

wpf layout, controls and tools

INTRODUCTION

Windows Presentation Foundation is the new, vector-based user interface system used by modern Windows applications within the .NET 3.5 Framework. This module takes a look at WPF and some of the tools you can use to create applications that use WPF.

OUTCOMES

By the end of this module, you should be able to do the following:

- ***Develop a basic WPF application.***
- ***Explain the parts of a WPF application including XAML definitions and the code-behind file.***
- ***Understand WPF layout panels.***

module 14: wpf layout, controls and tools

14.1 – Graphics platforms

The two traditional ways to implement graphics on the Windows platform are either **GDI** or **DirectX**.

GDI is a technology that is over 20 years old. It was created in an era when memory was very low, and processing power was just as scarce. Due to these constraints GDI was built to allow programmers to *paint* onto the screen. In this way no screen objects needed to be held in memory, instead the programmer would describe how to draw everything on the screen, and this would get called as needed.

The main problem with this approach is that a significant amount of code is required just to display all the screen elements, and an even greater amount of code is needed in order to enable any graphical effects like zooming, sliding items around in an animation, etc. Another major downside is that GDI is very resolution dependant. Trying to scale up to larger screens, or down to smaller PDA-like screens morphs things in strange ways.

DirectX is an API that allows developers to create very high performance two and three dimensional applications. It takes advantage of the graphics acceleration available on modern graphics cards. DirectX is more powerful for the developer as they no longer need to write the low level code to get pixels onto the screen. This is handled by DirectX.

However they still need to do a lot of the hard work of defining everything that can be displayed on the screen, positioning and laying everything out.

14.2 – What is WPF?

Windows Presentation Foundation (WPF) is a new umbrella technology which takes many types of presentation technologies and integrates them all into one managed API. Previously with the different Microsoft APIs it was troublesome to use them together in one application. Trying to have a 3D application which had support for media was difficult, or a Windows Form application that had controls and documents embedded in it usually required embedding external controls (like an IE browser) to view documents.

WPF combines all types of **presentation** technologies into one library (notice the word presentation and not graphics) enabling the creation rich applications. Now

you can create an application which uses layout to automatically position elements while incorporating 3D graphs and animated buttons. It includes:

- Controls
- Data binding
- Layout
- 2-D and 3-D graphics
- Animation
- Styles
- Templates
- Media
- Documents, typography

14.3 – The technology of WPF

WPF is a new technology that brings together and takes advantage of the improvements in hardware technology while utilising the new software advancements Microsoft has made.

WPF makes use of managed code in .NET allowing it to use all the features that come inherently with it such as ease of development and garbage collection but is built on top of DirectX and so can take advantage of hardware acceleration and will benefit from any performance improvements made later by the DirectX team.

As mentioned previously, GDI and other graphical libraries were focused around how to 'paint' the UI, for example "draw a red line, draw an ellipse". Due to increases in memory storage it is now more efficient for WPF to handle things from the other side "I have a red line, I have an ellipse".

Then the rendering technologies handle the complexities of drawing it on screen. This technique is extremely useful when trying to do advanced things with the UI like animation. Previously the programmer would be required to calculate how to move pixels around the screen, now WPF simplifies it by stating that "the red line is here, it has just moved to the right. Figure out how to draw it now".

WPF is resolution independent which is great in this age of differing screen sizes. Rather than the old corporate view of everyone with the same resolution CRT, today's work environment often spans a wide range of resolutions – from 20+" LCDs with resolutions of 1920x1600 or higher all the way down to Windows Mobile devices.

WPF utilises vectors for its graphics which easily allows the screen to be resized and still be crisp and clean looking. It also makes creating the layout of user

interfaces easier with the new layout panels, which also incorporate layout algorithms that keep things looking good when the window resizes.

But perhaps the most noticeable change is in the additional graphical features that can be used. Anti-aliasing is available automatically to all machines running Vista, buttons and other controls can be controlled with application wide styling and can be prettied with gradients, etc.

Animation is also incorporated into WPF allowing you to slide controls in/out, animate on mouse over, etc. all by simply defining start/end properties for attributes.

14.3.1 – WPF Tiers

Because WPF is built on top of DirectX, it gets its rendering capabilities from it and the power of the graphics card in the user's computer. Most video cards in today's laptops and PCs can view WPF applications at Tier 2, but older computers with integrated graphics may view applications rendered with Tier1 capabilities.

- Tier 2 Hardware Rendering
  - DirectX 9.0, Pixel Shaders 2.0, 128mb Video Ram
- Tier 1 Mixed Hardware/Software Rendering
  - DirectX 7.0, 32mb Video Ram
- Tier 0 Software Rendering
  - Exceptionally old video cards

14.4 – WPF Separates Form & Function

Screens in WPF are defined in **XAML** (e**X**tensible **A**pplication **M**ark-up **L**anguage). Designers can use XAML to create screens independently of the programmers allowing for parallel development. All screen layout, button styles, data binding, animation, etc. can be defined in XAML files. XAML can be edited using a variety of tools.

While XAML defines the front end, you still have a code behind file for all the application logic. This separation is similar to how ASP.NET handles .aspx code behind files.

14.5 – Tools

XAMLPad can be downloaded for free and allows you to create XAML and see instantly what it looks like. It's a great resource that allows you to play with XAML without needing to compile the application every time.

A more robust option is **Expression**. Microsoft has released a new suite of tools under the Expression umbrella. **Expression Design** is similar to Adobe Illustrator and is used to create single art assets. **Expression Blend** is used to create user interface screens. These tools are centred on the designer, and allows them to create the XAML independently of the programmer.

Finally, the Visual Studio designer can be used to create basic user interfaces.

14.6 – Control Elements

Controls can be nested inside each other in a hierarchy, this can allow for a deep level of screen customisation. For example, a ListBox could have buttons inside with each having a picture nested inside of it, something very difficult to achieve in non-WPF applications. Some of the controls available to the WPF application developer are:

- Standard
  - Button
  - Textbox / Label
  - Image
  - RadioButton / Checkbox
  - ListBox
- Media element
  - Embedded movies/streaming movies
- Viewport
  - Mapping 3D to the screen
- Layout (next slide)
- Controls can be nested inside each other
  - ListBox with a button inside with a picture nested inside of it.

14.7 – Layout

XAML assists in creating the layout of applications by supporting dynamic resizing of components. There are a number of different layout algorithms that can be used in applications, each allowing for dynamic resizing of an application with the controls reacting accordingly.

- **StackPanel** places all the controls in a stack, one on top of the other. It can be set to stack horizontally if desired.
- **DockPanel** is similar to how you dock controls in standard windows applications.
- **WrapPanel** places each control one after the other across the screen, and then wraps and starts placing them on the next 'row' underneath.

- **Grid** allows you to define your own columns & rows relatively to place your controls
- **Canvas** allows for absolute positioning of items.

Controls can be nested inside each other. As an example a ListBox could have buttons inside with each having a picture nested inside of it. Or a DockPanel hosting a StackPanel which in turn is hosting videos.

14.8 – Styles

XAML gives you the ability to theme and style your entire application easily. You can define application wide defaults for buttons, list boxes, etc. You can start off with everything a matte lime green, and then decide to make everything glossy peach. This is done easily by changing the XAML, no code changes are required.

It is similar to CSS in how you can set the style piece by piece, and use inheritance to build on top of styles already defined.

14.9 – Templates

Styles allow for changing the style of controls, font size, colour, etc. Templates allow you to override how a control renders itself. A button has a placeholder to put its content (usually text), however we may want to create a new style of button that has an image on the left hand side and its text on the right hand side.

Traditionally this required the creation of a brand new custom control class. XAML allows you to define a new template for the button and define what should be created inside of it, saving the need to get into code and writing a brand new class.

14.10 – WPF applications

WPF is used in two main ways:

1. The standard way is to develop a Windows application. This will involve the user having the executable on their computer and the .NET 3.0 Framework installed. A standard executable gives the most control to the developer as they can access the system as any standard application can.
2. The second choice is to deploy the application through a web page.
   The user goes to the page and the application will start running once it has been downloaded. The user is still required to have the full .NET 3.0 Framework installed but they do not need to store the executable on their computer; it is only held in the Internet cache temporarily.
   A downside to this is the security is tightly strapped down, and the developer will not be able to access anything on the local computer.

14.10.1 – Silverlight

Silverlight uses a subset of WPF and allows for cross platform/cross browser development of .NET powered Rich Internet Applications. It works on Windows and Mac and can be installed to work in Internet Explorer, FireFox, Opera and Safari. A Linux port is in development at the moment.

module 14: hands on lab

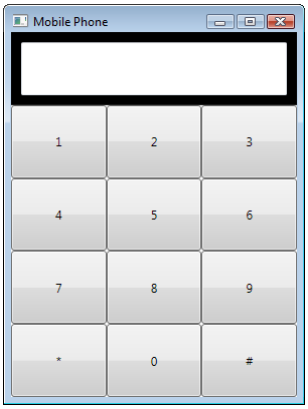
Exercise 1

This exercise will take you through the process of creating a WPF application, from concept to running.

You will create a basic mobile phone keypad that simply displays the number pressed. This exercise will give high level guidance, it is up to you to think and explore the tools.

Designing the interface

The application will mimic a very basic keypad and LCD. There will be an LCD text field at the top of the screen, accompanied by a 3x4 keypad. The screen will look something like this:



Think about how you could structure the hierarchy of controls to achieve this effect. Which of the types of layout WPF offers is best suited for this interface?

- A **StackPanel**? Perhaps you could do each of the button columns as a stack panel component.
- A **DockPanel**? That could get tedious trying to get the correct effect you're after.
- A **Grid**? This will fit your purpose perfectly. You can use the grid to define your own columns and rows. This allows you to place a button in each cell of the grid, with the LCD screen spanning the three columns in the first row.

To achieve the desired textbox appearance, you can embed the TextBox inside a DockPanel to position it just right.

Control hierarchy

This means that our hierarchy of controls will look something like this

- Grid
  - DockPanel
    - TextBox
  - Button (Key 1)
  - Button (Key 2)
  - Button (Key 3)
  - …

Special Note:

*Hint: When creating grid rows and columns, you can specify the width as * which works in the same way as it does in HTML tables – all * widths will get an equal amount of the remaining space.*

Creating the XAML

1. Create a new WPF C# application using either Expression Blend or Visual Studio.
2. Use the editor in whichever application you like to create the XAML. Spend some time playing around trying to get the grid correct.
   The following XAML is a sample of how your form can be designed. Note that yours may vary due to extra attributes or deciding to use a different layout.

Special Note:

Hint: To have the buttons take up the entire size of the cell they're in, remove the Margin, Width and Height properties.

```
<Window xmlns:my3="clr-
namespace:System.Windows.Controls;assembly=PresentationFramework"
xmlns:my2="clr-
namespace:System.Windows.Controls;assembly=PresentationFramework"
xmlns:my1="clr-
namespace:System.Windows.Controls;assembly=PresentationFramework"
xmlns:my="clr-
namespace:System.Windows.Controls;assembly=PresentationFramework"
x:Class="PhoneApp.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Mobile Phone" Height="400" Width="300">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="*" />
                <RowDefinition Height="*" />
                <RowDefinition Height="*" />
                <RowDefinition Height="*" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>

            <DockPanel Grid.Row="0" Grid.Column="0"
                       Grid.ColumnSpan="3" Grid.RowSpan="1"
                       Background="Black">
                <TextBox Margin="10" Name="LCD" />
            </DockPanel>
            <Button Grid.Row="1" Grid.Column="0">1</Button>
            <Button Grid.Row="1" Grid.Column="1">2</Button>
            <Button Grid.Row="1" Grid.Column="2">3</Button>
            <Button Grid.Row="2" Grid.Column="0">4</Button>
            <Button Grid.Row="2" Grid.Column="1">5</Button>
            <Button Grid.Row="2" Grid.Column="2">6</Button>
            <Button Grid.Row="3" Grid.Column="0">7</Button>
            <Button Grid.Row="3" Grid.Column="1">8</Button>
```

```
            <Button Grid.Row="3" Grid.Column="2">9</Button>
            <Button Grid.Row="4" Grid.Column="0">*</Button>
            <Button Grid.Row="4" Grid.Column="1">0</Button>
            <Button Grid.Row="4" Grid.Column="2">#</Button>

        </Grid>
</Window>
```

Wiring events

We are going to be writing some code, so switch to Visual Studio.

1. Double click on a keypad button to create a **Click** event.
2. Write a method that will add to the LCD display the number on the keypad that was pressed. This could be done by creating an individual event for each button, or we can create a more generic one that just adds the text that is on the button:

```
private void Button_Click(object sender,
    RoutedEventArgs e)
{
    Button clickedbutton = (Button)sender;
    LCD.Text += clickedbutton.Content;
}
```

3. Wire up each of the other buttons to call this method.
4. Run the application. Each button press should add the number to the LCD, like a mobile phone.