Windows Presentation Foundation (WPF)

Windows Programming Course

Agenda – Windows Presentation Foundation

1. WPF Introduction
2. WPF Layout
3. WPF Controls
4. Databinding & MVVM Commands
5. Styles, Triggers and Control Templates
6. Web Service
7. Entity Framework

Agenda

1. WPF Introduction
2. WPF Architecture
3. XAML Markup
4. WPF User Interface: Layout and UI Tree

1.
WPF Introduction

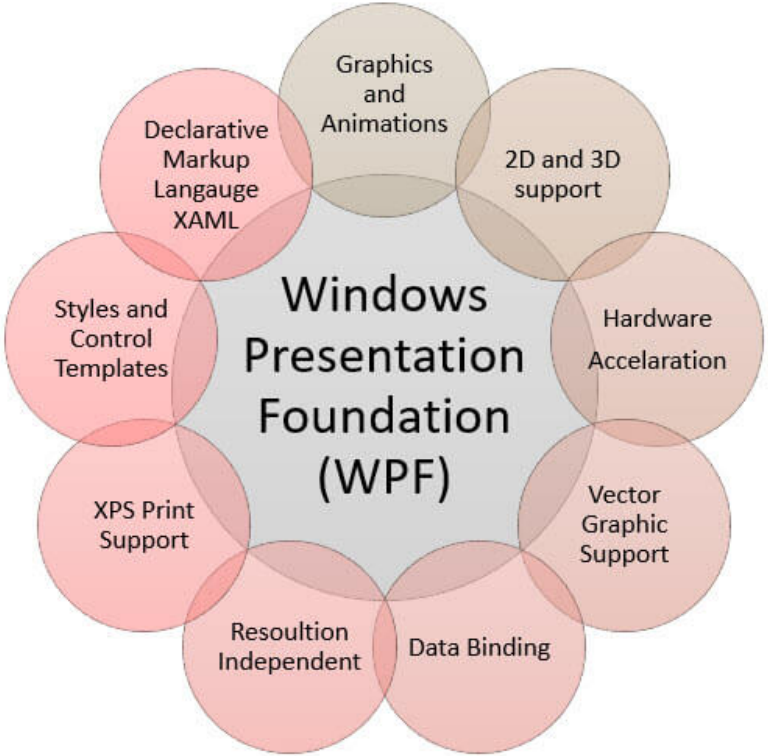
Windows Presentation Foundation (WPF)

WPF is the new graphical subsystem in Windows Operating System that built with .NET Framework.

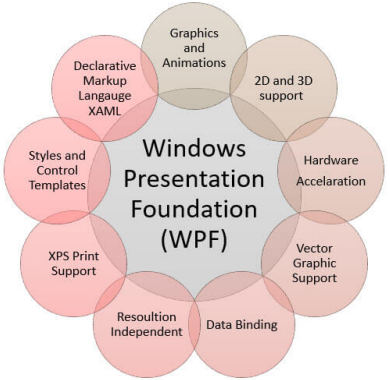
It is a collection of class libraries that comes with all the latest versions of .NET framework for creating and executing rich and interactive User Interface (Presentation).

Developer tools: Visual Studio, Expression Studio.

Features


Graphics and Animations
2D and 3D support
Hardware Accelaration
Vector Graphic Support
Data Binding
Resoultion Independent
XPS Print Support
Styles and Control Templates
Declarative Markup Langauge XAML
Windows Presentation Foundation (WPF)

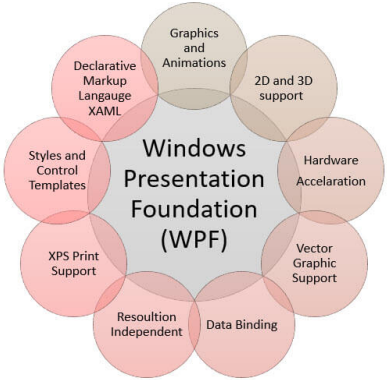
Features (cont.)



All WPF Applications are **DirectX based**.

- WPF uses DirectX for media rendering where old Windows Forms Application uses GDI+ (Graphics Device Interface).
- So it offloads works to GPU (Graphics Processing Unit) instead of CPU.
- The result is high quality rich media UI and resolution independence.

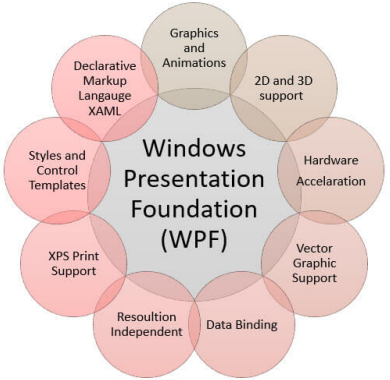
Features (cont.)



Separation of Concerns: There is two parts appearance of UI and its behavior.

- Appearance means to say application User Interface and appearance is specified by XAML.
- Behavior means to say how the application work and it is handled by .NET language like C#/VB.Net, etc.

Features (cont.)



Data binding

- Mechanism to display and interact with data between UI elements and data object on user interface.

Templates

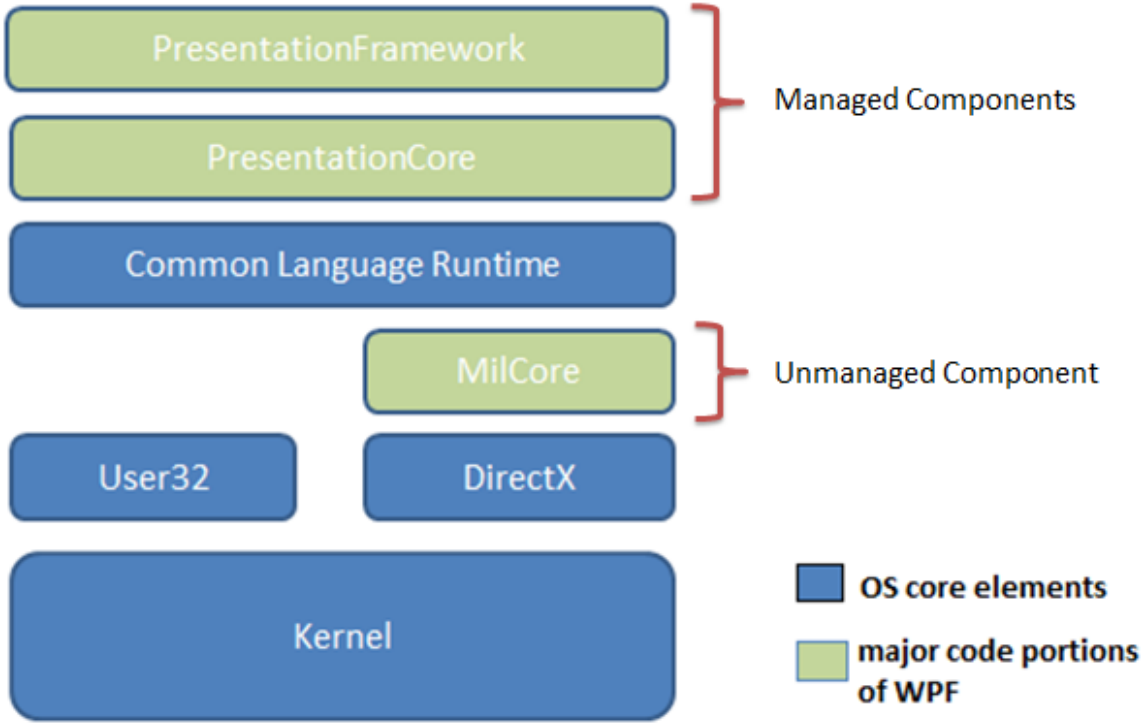
- In WPF you can define the look of an element directly with a Template

Animations

- Building interactivity and movement on user Interface

2.
WPF Architecture

WPF Architecture



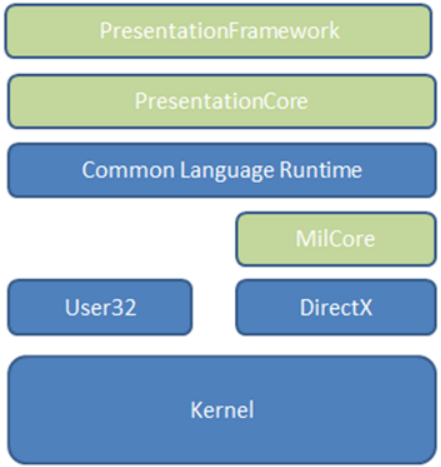
WPF Architecture (cont.)

Managed Components:

- **Presentation Framework** provide many high level elements like Controls (Textbox, Button etc), Styles, Layout, Templates, Binding etc.
- **Presentation Core** contains basic class like UIElement, Visual. Visual is a class that provide rendering support in WPF.

Unmanaged Component:

- **MilCore** offers tight integration with DirectX



Markup and Code-behind

WPF Application are made up of **markup (XAML)** and **code-behind** (managed code)

- The markup define the layout and appearance of the application
- The code-behind defines the behavior

Markup example

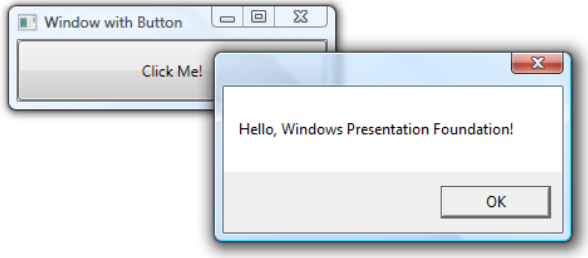
```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>
</Window>
```

Code-behind example

```
using System.Windows; // Window, RoutedEventArgs, MessageBox
namespace SDKSample {
    public partial class AWindow : Window {
        public AWindow() {
            // InitializeComponent call is required to merge the UI
            // that is defined in markup with this class, including
            // setting properties and registering event handlers
            InitializeComponent();
        }

        void button_Click(object sender, RoutedEventArgs e) {
            // Show message box when button is clicked.
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```



3.
XAML Markup

What is XAML?

XAML stands for *eXtensible Application Markup Language*.

XAML is a declarative language for building UI

- It describes the behavior and integration of components (in most cases UI components)
- Cannot describe business logic.

Declarative style

```
<Button Content="Click me!"/>
```

Programmatical style

```
Button button = new Button();
button.Content = "Click me!"
```

Sample XAML code

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Window1"
    x:Name="Window"
    Title="SimpleCritters"
    Width="557" Height="156"
    FontSize="22" FontWeight="Bold" FontFamily="Comic Sans MS"
    >
    <StackPanel Orientation="Horizontal" Margin="5">
        <StackPanel>
            <Image Margin="5" Height="50" Source="Frog.jpg"/>
            <Label Margin="5" Content="Frog"/>
        </StackPanel>
        <StackPanel>
            <Image Margin="5" Height="50" Source="Butterfly.jpg"/>
            <Label Margin="5" Content="Butterfly"/>
        </StackPanel>
        <StackPanel>
            <Image Margin="5" Height="50" Source="Shark.jpg"/>
            <Label Margin="5" Content="Shark"/>
        </StackPanel>
        <StackPanel>
            <Image Margin="5" Height="50" Source="Tiger.jpg"/>
            <Label Margin="5" Content="Tiger"/>
        </StackPanel>
        <StackPanel>
            <Image Margin="5" Height="50" Source="Platypus.jpg"/>
            <Label Margin="5" Content="Platypus"/>
        </StackPanel>
    </StackPanel>
</Window>
```

The variants of XAML

- **WPF XAML** encompasses the elements that describe WPF content (WPF Application), such as vector graphics, controls, and documents
- **XPS XAML** is the part of WPF XAML that defines an XML representation for formatted electronic documents.
- **Silverlight XAML** is a subset of WPF XAML that's intended for Microsoft Silverlight applications. Silverlight is a cross-platform browser plug-in that allows you to create rich web content with two-dimensional graphics, animation, and audio and video.
- **Xamarin.Forms XAML** defines user interfaces using all the Xamarin.Forms views, layouts, and pages, as well as custom classes.

XAML Basic Rules

- Every element in a XAML document maps to an instance of a .NET class. The name of the element matches the name of the class exactly.
- As with any XML document, you can nest one element inside another.
- You can set the properties of each class through attributes.

XAML Namespaces

It’s not enough to supply just a class name. The XAML parser also needs to know the .NET namespace where this class is located.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Window1"
    x:Name="Window"
    Title="SimpleCritters"
    Width="557" Height="156"
    FontSize="22" FontWeight="Bold" FontFamily="Comic Sans MS"
    >
```

- `http://schemas.microsoft.com/winfx/2006/xaml/presentation` is the core WPF namespace. It encompasses all the WPF classes, including the controls you use to build user interfaces.
- `http://schemas.microsoft.com/winfx/2006/xaml` is the XAML namespace. It includes various XAML utility features that allow you to influence how your document is interpreted. This namespace is mapped to the prefix x. Usage example: <x:ElementName>

Using Types for Other Namespaces

```
xmlns:Prefix="clr-namespace:Namespace;assembly=AssemblyName”
```

Prefix: This is the XML prefix you want to use to indicate that namespace in your XAML markup. For example, the XAML language uses the x prefix.

Namespace: This is the fully qualified .NET namespace name.

AssemblyName: This is the assembly where the type is declared, without the .dll extension. This assembly must be referenced in your project.

Naming Elements

You can set either the XAML Name property (using the x prefix) or the Name property that belongs to the actual element (by leaving out the prefix).

```
<Button Name="btnClick"></Button>

<Grid x:Name="grid1">
</Grid>
```

Properties in XAML

- **Attribute syntax** is the XAML markup syntax that sets a value for a property by declaring an attribute on an existing object element.

```
<TextBox Name="txtQuestion"
  VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
  FontFamily="Verdana" FontSize="24" Foreground="Green" ...>
```

Properties in XAML (cont.)

- **Property element syntax** <Typename.PropertyName>

```
<Button>
  <Button.Background>
    <SolidColorBrush Color="Blue"/>
  </Button.Background>
  <Button.Content>
    This is a button
  </Button.Content>
</Button>
```

Properties in XAML (cont.)

- **Attached Properties** are properties that may apply to several controls but are defined in a different class, syntax: `DefiningType.PropertyName`.

```
<TextBox Grid.Row="0">
</TextBox>
```

Nesting Elements

```
<StackPanel Orientation="Horizontal" Margin="5">
    <StackPanel>
        <Image Margin="5" Height="50" Source="Frog.jpg"/>
        <Label Margin="5" Content="Frog"/>
    </StackPanel>
    <StackPanel>
        <Image Margin="5" Height="50" Source="Butterfly.jpg"/>
        <Label Margin="5" Content="Butterfly"/>
    </StackPanel>
    <StackPanel>
        <Image Margin="5" Height="50" Source="Shark.jpg"/>
        <Label Margin="5" Content="Shark"/>
    </StackPanel>
    <StackPanel>
        <Image Margin="5" Height="50" Source="Tiger.jpg"/>
        <Label Margin="5" Content="Tiger"/>
    </StackPanel>
    <StackPanel>
        <Image Margin="5" Height="50" Source="Platypus.jpg"/>
        <Label Margin="5" Content="Platypus"/>
    </StackPanel>
</StackPanel>
```

Object Tree

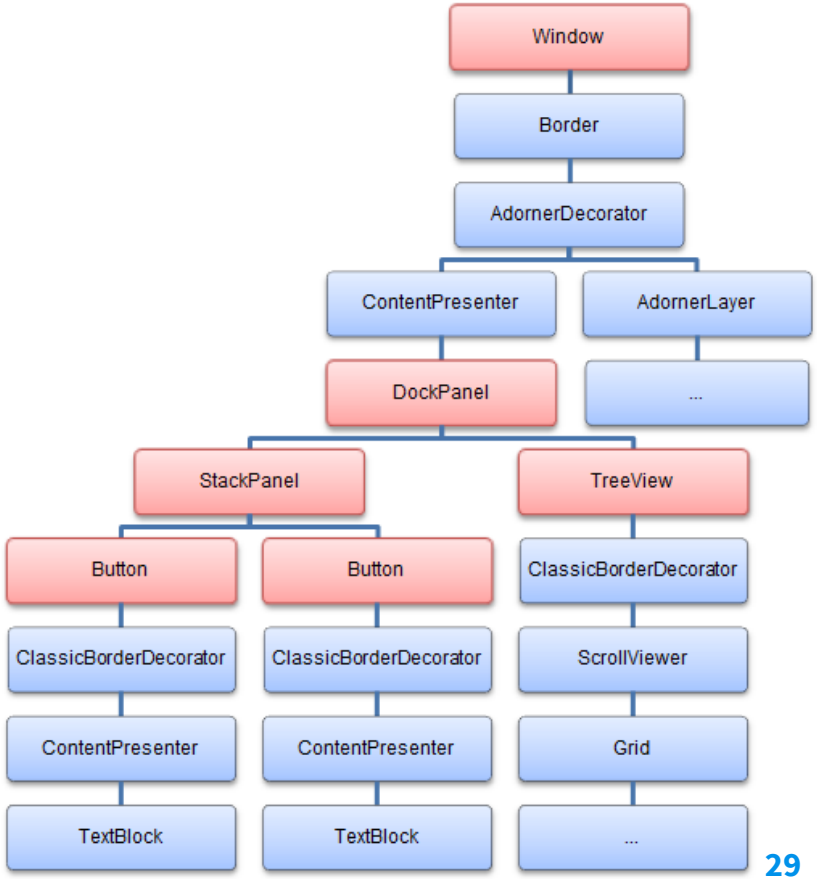
Logical Tree is defined by the content relationships among the objects in the interface. That includes controls contained within other controls and simple content.

Visual Tree represents the structure of visual objects including the components that define them.

Object Tree (cont.)

Why should you care about the logical and visual trees?

- Controls tend to inherit property values according to their positions in the logical tree.
- Events tend to follow the visual tree.



Read more

- XAML Overview
- XAML Syntax in Detail

4.
WPF User Interface

Layout

WPF Key Principles

- *Elements (such as controls) should not be explicitly sized*. Instead, they grow to fit their content. For example, a button expands as you add more text. You can limit controls to acceptable sizes by setting a maximum and minimum size.

- *Elements do not indicate their position with screen coordinates*. Instead, they are arranged by their container based on their size, order, and (optionally) other information that's specific to the layout container.

WPF Key Principles (cont.)

- *Layout containers "share" the available space among their children.* They attempt to give each element its preferred size (based on its content) if the space is available. They can also distribute extra space to one or more children.

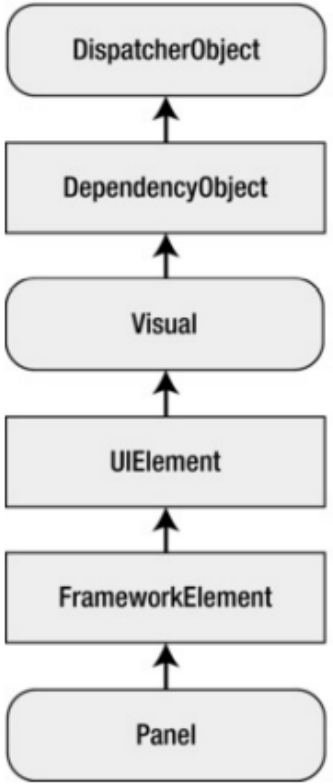
- *Layout containers can be nested.*

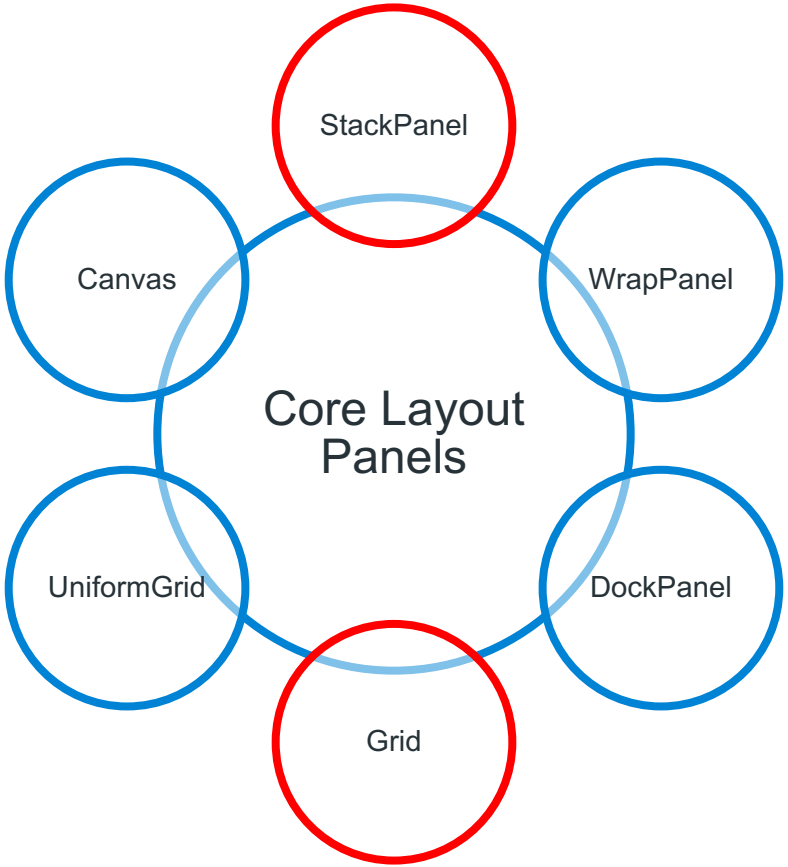
The abstract System.Windows.Controls.Panel class

All the WPF layout containers are panels that derive from the abstract `System.Windows.Controls.Panel` class.

Public Properties of the Panel Class:

Name	Description
Background	The brush that's used to paint the panel background
Children	The collection of items that's stored in the panel.

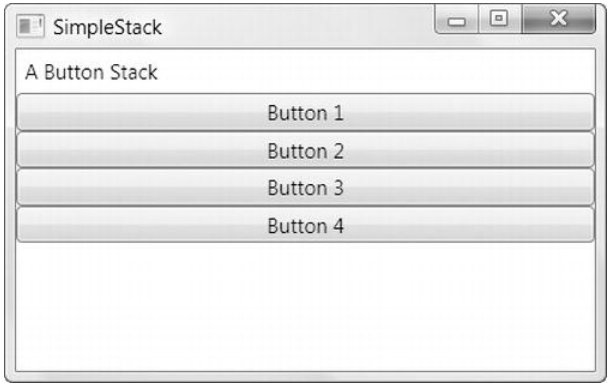



StackPanel
Canvas
WrapPanel
Core Layout Panels
UniformGrid
DockPanel
Grid

The StackPanel

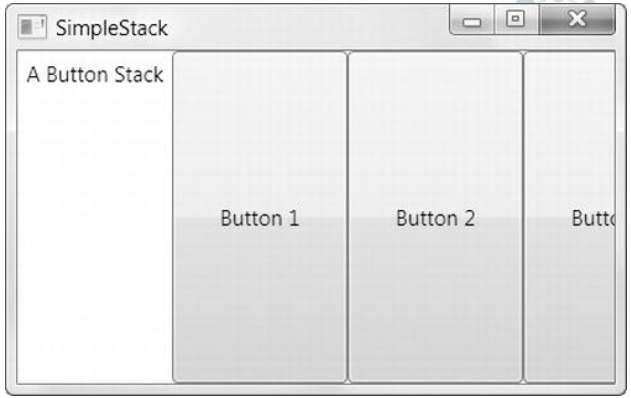
- Places elements in a horizontal or vertical stack. This layout container is typically used for small sections of a larger, more complex window.

```
<StackPanel>
  <Label>A Button Stack</Label>
  <Button>Button 1</Button>
  <Button>Button 2</Button>
  <Button>Button 3</Button>
  <Button>Button 4</Button>
</StackPanel>
```



The StackPanel

```
<StackPanel Orientation="Horizontal">
  <Label>A Button Stack</Label>
  <Button>Button 1</Button>
  <Button>Button 2</Button>
  <Button>Button 3</Button>
  <Button>Button 4</Button>
</StackPanel>
```

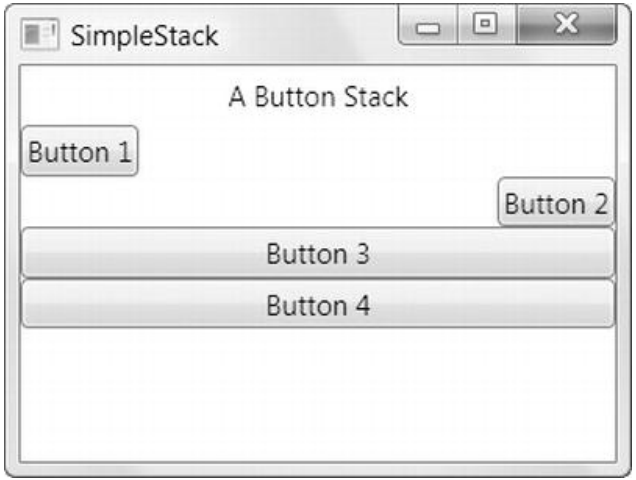


The StackPanel – Layout Properties

Name	Description
HorizontalAlignment	Determines how a child is positioned inside a layout container when there's extra horizontal space available. You can choose Center, Left, Right, or Stretch.
VerticalAlignment	Determines how a child is positioned inside a layout container when there's extra vertical space available. You can choose Center, Top, Bottom, or Stretch.
Margin	Adds a bit of breathing room around an element. The Margin property is an instance of the System.Windows.Thickness structure, with separate components for the top, bottom, left, and right edges.
MinWidth and MinHeight	Sets the minimum dimensions of an element. If an element is too large for its layout container, it will be cropped to fit.
MaxWidth and MaxHeight	Sets the maximum dimensions of an element. If the container has more room available, the element won't be enlarged beyond these bounds, even if the HorizontalAlignment and VerticalAlignment properties are set to Stretch.
Width and Height	Explicitly sets the size of an element.

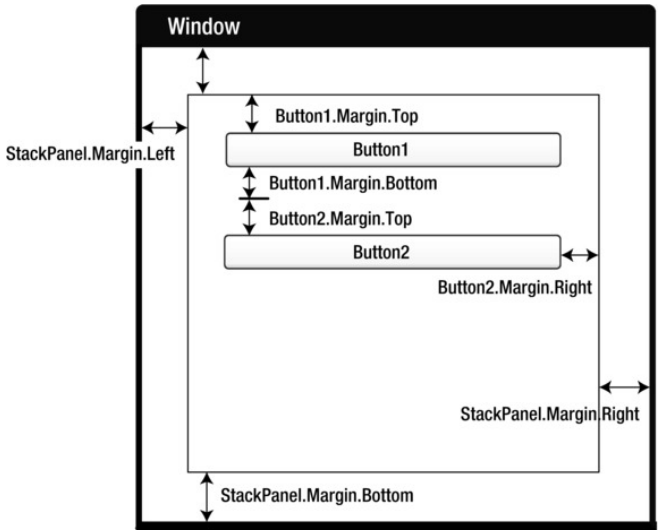
The StackPanel – Layout Properties – Alignment

```
<StackPanel>
  <Label HorizontalAlignment="Center">A Button Stack</Label>
  <Button HorizontalAlignment="Left">Button 1</Button>
  <Button HorizontalAlignment="Right">Button 2</Button>
  <Button>Button 3</Button>
  <Button>Button 4</Button>
</StackPanel>
```



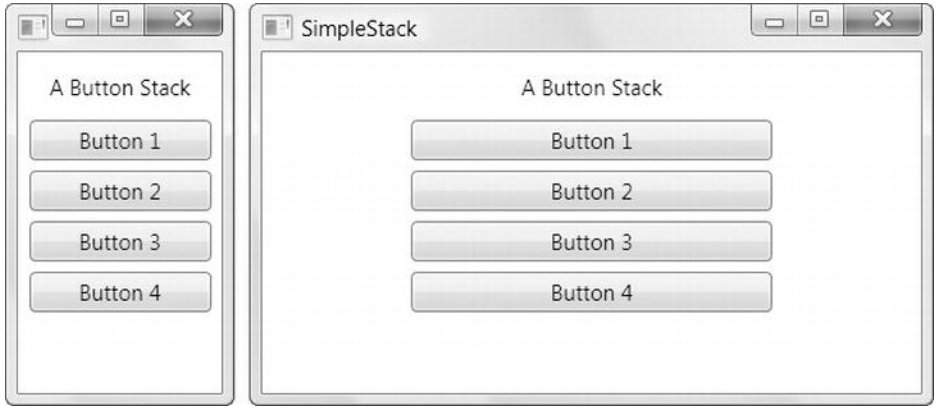
The StackPanel – Layout Properties – Margin

```
<StackPanel Margin="3">
  <Label Margin="3" HorizontalAlignment="Center">A Button Stack</Label>
  <Button Margin=”3” HorizontalAlignment="Left">Button 1</Button>
  <Button Margin=”3” HorizontalAlignment="Right">Button 2</Button>
  <Button Margin=”3”>Button 3</Button>
  <Button Margin=”3”>Button 4</Button>
</StackPanel>
```



The StackPanel – Layout Properties – Minimum, Maximum, and Explicit Sizes

```
<StackPanel Margin="3">
  <Label Margin="3" HorizontalAlignment="Center">A Button Stack</Label>
  <Button Margin="3" MaxWidth="200" MinWidth="100">Button 1</Button>
  <Button Margin="3" MaxWidth="200" MinWidth="100">Button 2</Button>
  <Button Margin="3" MaxWidth="200" MinWidth="100">Button 3</Button>
  <Button Margin="3" MaxWidth="200" MinWidth="100">Button 4</Button>
</StackPanel>
```

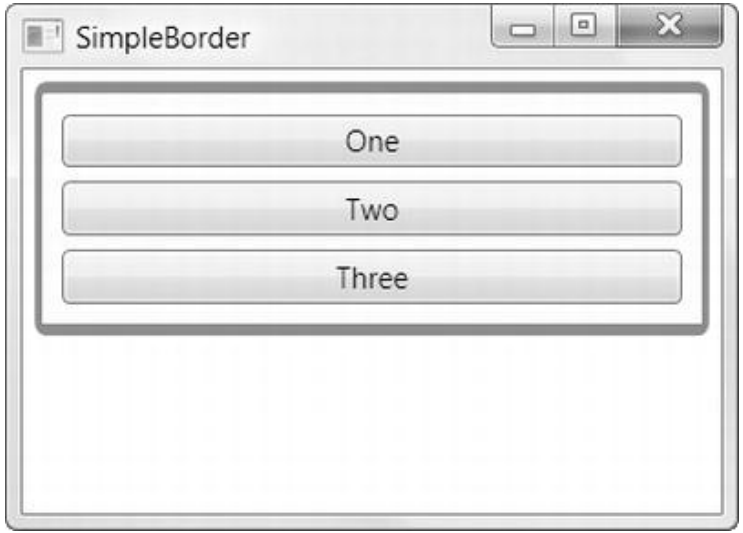


The StackPanel – The Border

Name	Description
Background	Sets a background that appears behind all the content in the border by using a Brush object
BorderBrush and BorderThickness	Sets the color of the border that appears at the edge of the Border object, using a Brush object, and sets the width of the border, respectively.
CornerRadius	Allows you to gracefully round the corners of your border.
Padding	Adds spacing between the border and the content inside.

The StackPanel – The Border

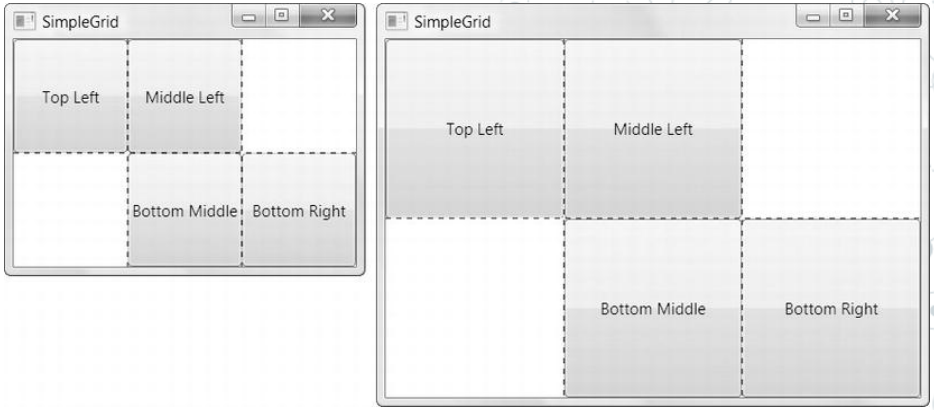
```
<Border Margin="5" Padding="5" Background="LightYellow"
BorderBrush="SteelBlue" BorderThickness="3,5,3,5" CornerRadius="3"
VerticalAlignment="Top">
  <StackPanel>
    <Button Margin="3">One</Button>
    <Button Margin="3">Two</Button>
    <Button Margin="3">Three</Button>
  </StackPanel>
</Border>
```



The Grid

- Arranging elements in rows and columns according to an invisible table. This is one of the most flexible and commonly used layout containers.
- Carving your window into smaller regions that you can manage with other panels.
- Creating a Grid-based layout is a two-step process.
  - Choose the number of columns and rows that you want.
  - Assign the appropriate row and column to each contained element, thereby placing it in just the right spot.

The Grid – Creating




```
<Grid ShowGridLines="True">
<Grid.RowDefinitions>
  <RowDefinition></RowDefinition>
  <RowDefinition></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>
  <Button Grid.Row="0" Grid.Column="0">Top Left</Button>
  <Button Grid.Row="0" Grid.Column="1">Middle Left</Button>
  <Button Grid.Row="1" Grid.Column="2">Bottom Right</Button>
  <Button Grid.Row="1" Grid.Column="1">Bottom Middle</Button>
</Grid>
```

The Grid – Fine-Tuning Rows and Columns

The Grid supports three sizing strategies:

- **Absolute sizes**: the exact size by using device-independent units. This is the least useful strategy because it's not flexible enough to deal with changing content size, changing container size, or localization.

```
<ColumnDefinition Width="100"></ColumnDefinition>
```

- **Automatic sizes**: Each row or column is given exactly the amount of space it needs, and no more. This is one of the most useful sizing modes.

```
<ColumnDefinition Width="Auto"></ColumnDefinition>
```

- **Proportional sizes**: Space is divided between a group of rows or columns. This is the standard setting for all rows and columns.

```
<ColumnDefinition Width="*"></ColumnDefinition>
<ColumnDefinition Width=”2*"></ColumnDefinition>
```

The Grid – Spanning Rows and Columns

Using two more attached properties to make an element stretch over several cells: RowSpan and ColumnSpan.

E.g.: This button will take all the space that's available in the first and second cell of the first row:

```
<Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2">Span Button</Button>
```

And this button will stretch over four cells in total by spanning two columns and two rows:

```
<Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" Grid.ColumnSpan="2"> Span Button</Button>
```

The WrapPanel

The WrapPanel lays out controls in the available space, one line or column at a time.

By default, the **WrapPanel.Orientation** property is set to Horizontal; controls are arranged from left to right and then on subsequent rows.

The WrapPanel (cont.)

```
<WrapPanel Margin="3">
  <Button VerticalAlignment="Top">Top Button</Button>
  <Button MinHeight="60">Tall Button 2</Button>
  <Button VerticalAlignment="Bottom">Bottom Button</Button>
  <Button>Stretch Button</Button>
  <Button VerticalAlignment="Center">Centered Button</Button>
</WrapPanel>
```

The DockPanel

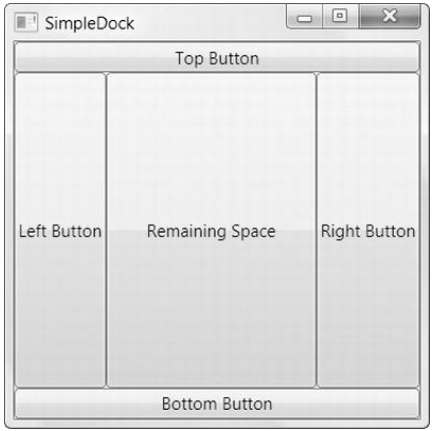
The DockPanel stretches controls against one of its outside edges.

The easiest way to visualize this is to think of the toolbars that sit at the top of many Windows applications. These toolbars are docked to the top of the window.

`LastChildFill="True"`: The DockPanel gives the remaining space to the last element.

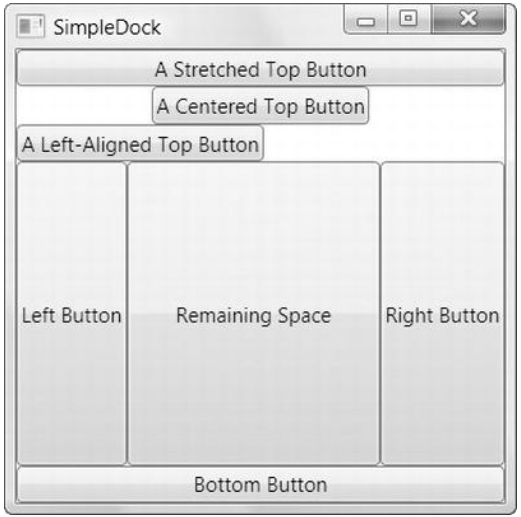
The DockPanel (cont.)

```
<DockPanel LastChildFill="True">
  <Button DockPanel.Dock="Top">Top Button</Button>
  <Button DockPanel.Dock="Bottom">Bottom Button</Button>
  <Button DockPanel.Dock="Left">Left Button</Button>
  <Button DockPanel.Dock="Right">Right Button</Button>
  <Button>Remaining Space</Button>
</DockPanel>
```



The DockPanel (cont.)

```
<DockPanel LastChildFill="True">
  <Button DockPanel.Dock="Top">A Stretched Top Button</Button>
  <Button DockPanel.Dock="Top" HorizontalAlignment="Center">A Centered Top
Button</Button>
  <Button DockPanel.Dock="Top" HorizontalAlignment="Left">A Left-Aligned Top
Button</Button>
  <Button DockPanel.Dock="Bottom">Bottom Button</Button>
  <Button DockPanel.Dock="Left">Left Button</Button>
  <Button DockPanel.Dock="Right">Right Button</Button>
  <Button>Remaining Space</Button>
</DockPanel>
```



Thanks!

Any questions?

You can find me at:

tranminhphuoc@gmail.com