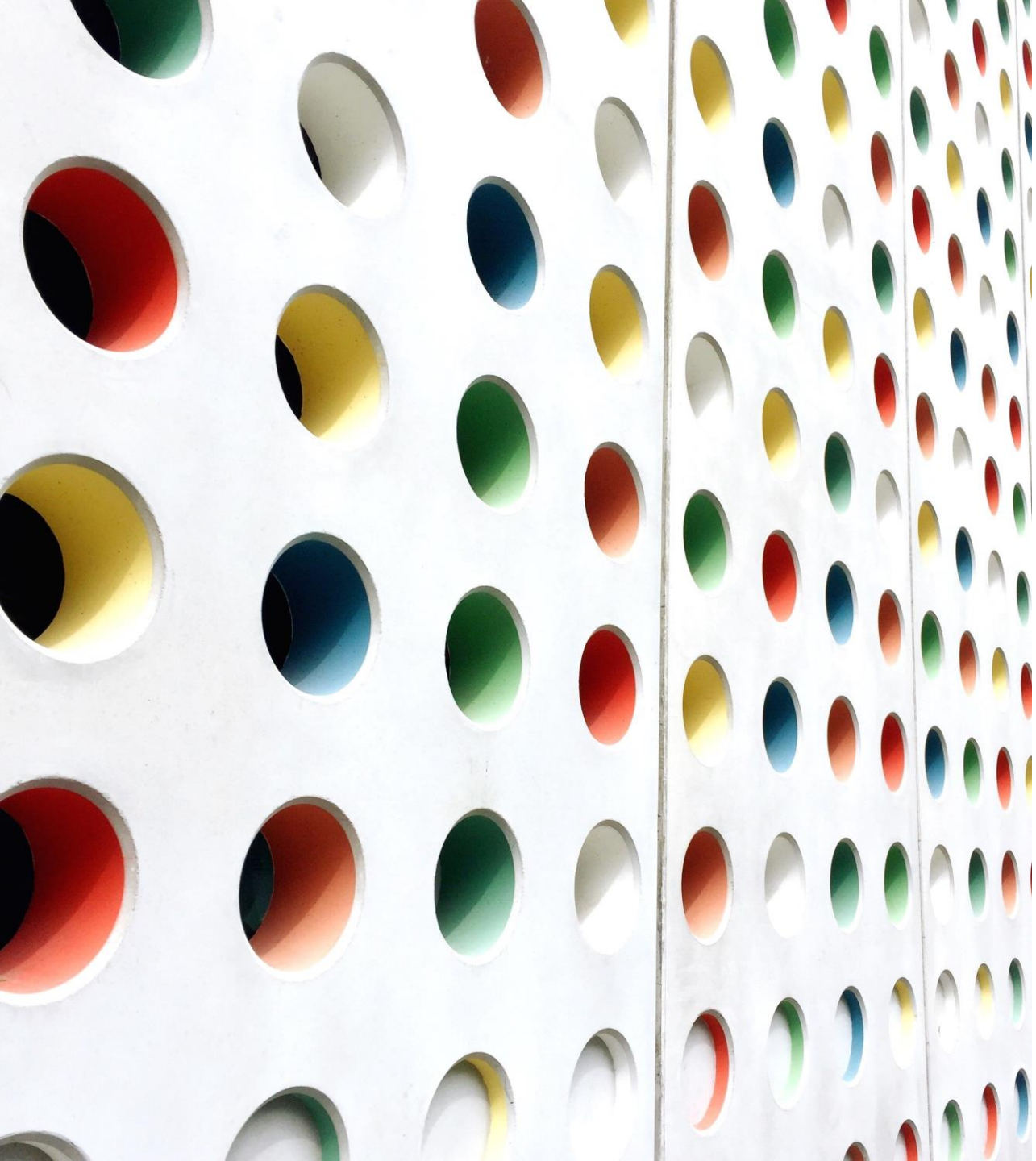
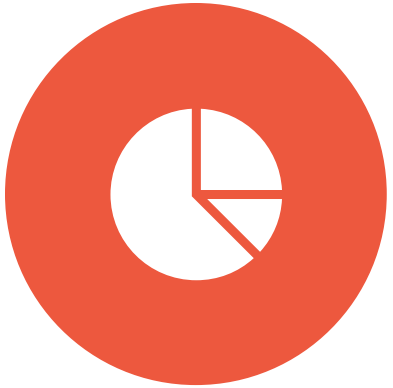


Windows Presentation Foundation (WPF)

ARCTECH INFO PRIVATE LIMITED

Title Lorem Ipsum



LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT.

NUNC VIVERRA IMPERDIET ENIM. FUSCE EST. VIVAMUS A TELLUS.

PELLENTESQUE HABITANT MORBI TRISTIQUE SENECTUS ET NETUS.

What is WPF

Windows Presentation Foundation is a UI framework that creates desktop client applications.

WPF is part of .NET, so if you have previously built applications with .NET using ASP.NET or Windows Forms, the programming experience should be familiar.

WPF uses the Extensible Application Markup Language (XAML) to provide a declarative model for application programming.

Program with WPF

WPF types are mostly located in the System.Windows namespace.

If you have previously built applications with .NET with frameworks like ASP.NET and Windows Forms, the fundamental WPF programming experience should be familiar, you:

- Instantiate classes
- Set properties
- Call methods
- Handle events

WPF includes more programming constructs that enhance properties and events

- dependency properties and
- routed events.

Markup and code-behind

WPF lets you develop an application using both markup and code-behind, an experience with which ASP.NET developers should be familiar.

You generally use XAML markup to implement the appearance of an application while using managed programming languages (code-behind) to implement its behavior.

This separation of appearance and behavior has the following benefits:

- Development and maintenance costs are reduced.
- Development is more efficient
- Globalization and localization for WPF applications is simplified.

Markup - XAML

XAML – eXtensible Application Markup Language, is Microsoft's implementation of XML for describing a GUI

You typically use it to define windows, dialog boxes, pages, and user controls, and to fill them with controls, shapes, and graphics.

The following example uses XAML to implement the appearance of a window that contains a single button:

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Week13Day01.MyFirstWindow“
    Title="Window with Button"
    Width="250" Height="100">

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

Code-behind

The main behavior of an application is to respond to user interactions.

For example, when user clicks a menu or button, invoke some business logic and data access logic in response.

In WPF, this behavior is implemented in a code-behind file that is associated with the xaml file.

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="HelloWorldApp.MyWindow"
Title="Window with Button"
Width="250" Height="100">
    <!-- Add button to window -->
    <Button Name="Button1" Click="Button1_Click">Click Me!</Button>
</Window>
```

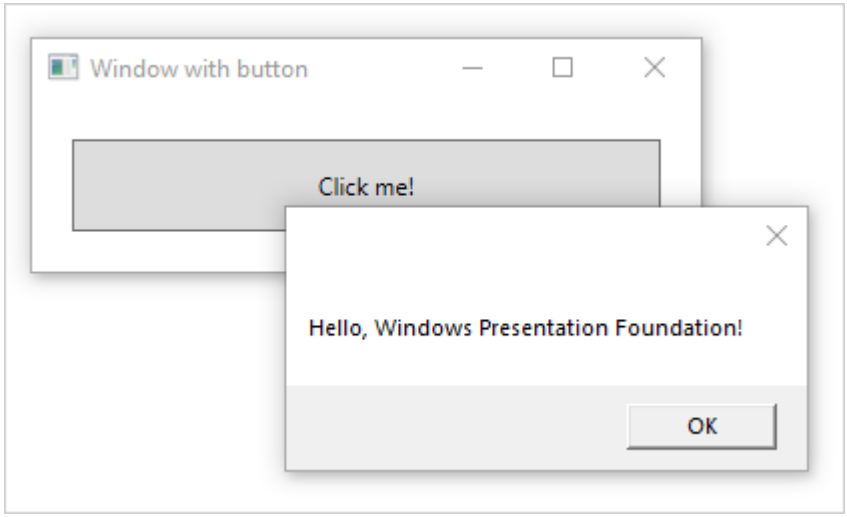
```
using System.Windows;
namespace HelloWorldApp
{
    public partial class MyWindow : Window
    {
        public MyWindow()
        {
            InitializeComponent();
        }

        void Button1_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```

Code Behind - InitalizeComponent

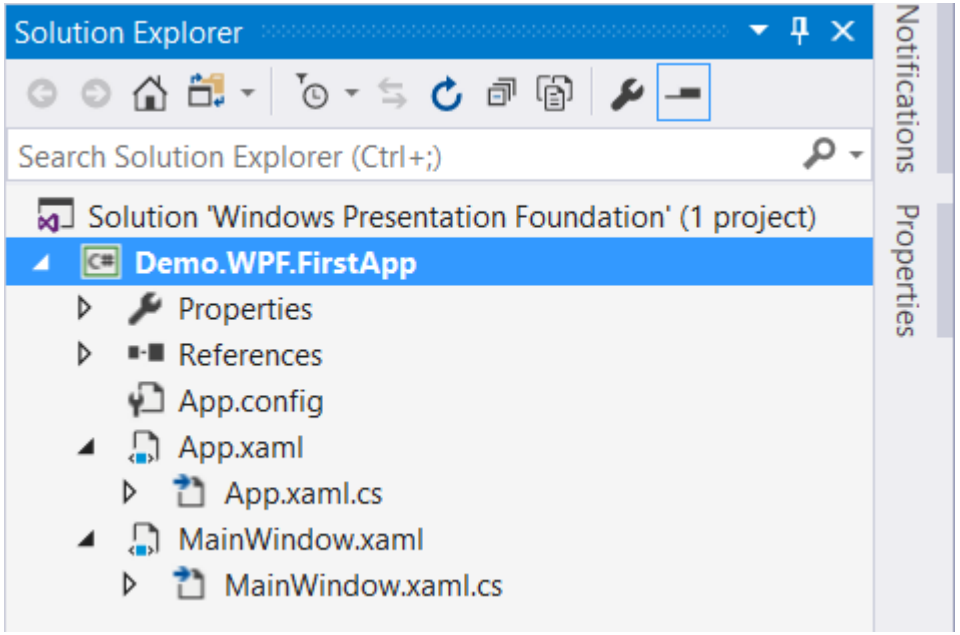
InitializeComponent is called from the code-behind class's constructor to merge the UI that is defined in markup with the code-behind class.

The combination of x:Class and InitializeComponent ensure that all elements include events are correctly implemented.

InitializeComponent has auto generated C# code based on the XAML.



WPF Project Structure



In a new WPF Project, 5 files are created automatically by Visual Studio

- App.config
  - Is used to store application settings like connectionstring etc.
- App.xaml & App.xaml.cs
  - Xaml is declarative start point of your application
  - Inherits Application class
  - Can subscribe to application-specific styles, events like `StartUp`, unhandled exceptions etc.
  - Xaml.cs is the code behind file
  - A project can have only one app.xaml file
- MainWindow.xaml & MainWindow.xaml.cs
  - Is the declarative window definition
  - Inherits Window class
  - Xaml.cs is the code behind file for the
  - A project can have multiple window xaml files

WPF Controls

The user experience is constructed using WPF controls. In WPF, control is an umbrella term that applies to a category of WPF classes that have the following characteristics:

- Hosted in either a window or a page.
- Have a user interface.
- Implement some behavior.

Built-in WPF controls

- **Buttons**: Button and RepeatButton.
- **Data Display**: DataGrid, ListView, and TreeView.
- **Date Display and Selection**: Calendar and DatePicker.
- **Dialog Boxes**: OpenFileDialog, PrintDialog, and SaveFileDialog.
- **Digital Ink**: InkCanvas and InkPresenter.
- **Documents**: DocumentViewer, FlowDocumentReader, FlowDocumentScrollViewer, and StickyNoteControl.
- **Input**: TextBox, RichTextBox, and PasswordBox.
- **Layout**: Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window, and WrapPanel.
- **Media**: Image, MediaElement, and SoundPlayerAction.
- **Menus**: ContextMenu, Menu, and ToolBar.
- **Navigation**: Frame, Hyperlink, Page, NavigationWindow, and TabControl.
- **Selection**: CheckBox, ComboBox, ListBox, RadioButton, and Slider.
- **User Information**: AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock, and ToolTip.

Layout

To create a user interface, arrange your controls by location and size to form a layout.

A key requirement of any layout is to adapt to changes in window size and display settings.

Rather than writing code to adapt a layout in these circumstances, WPF provides a first-class, extensible layout system for you.

The WPF layout system has relative positioning, which increases the ability to adapt to changing window and display conditions.

The layout system also manages the negotiation between controls to determine the layout. The negotiation is a two-step process:

- first, a control tells its parent what location and size it requires.
- Second, the parent tells the control what space it can have.

Canvas: Child controls provide their own layout.

DockPanel: Child controls are aligned to the edges of the panel.

Grid: Child controls are positioned by rows and columns.

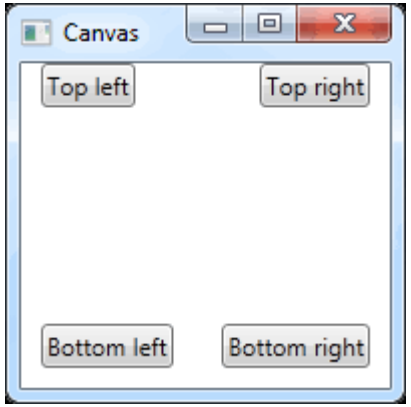
StackPanel: Child controls are stacked either vertically or horizontally.

VirtualizingStackPanel: Child controls are virtualized and arranged on a single line that is either horizontally or vertically oriented.

WrapPanel: Child controls are positioned in left-to-right order and wrapped to the next line when there isn't enough space. on the current line.

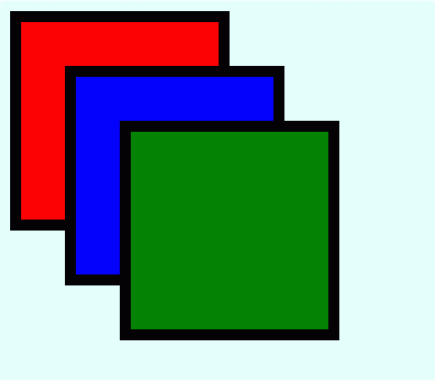
Layout - Canvas

Canvas panel is the basic layout Panel in which the child elements can be positioned explicitly using coordinates that are relative to the Canvas any side such as left, right, top and bottom




```
<Canvas>
    <Button Canvas.Left="10">Top left</Button>
    <Button Canvas.Right="10">Top right</Button>
    <Button Canvas.Left="10" Canvas.Bottom="10">Bottom left</Button>
    <Button Canvas.Right="10" Canvas.Bottom="10">Bottom right</Button>
</Canvas>
```

```
<Canvas Background="LightCyan">
    <Rectangle Canvas.Left="10" Canvas.Top="10" Height="200" Width="200" Stroke="Black"
               StrokeThickness="10" Fill="Red" />
    <Rectangle Canvas.Left="60" Canvas.Top="60" Height="200" Width="200" Stroke="Black"
               StrokeThickness="10" Fill="Blue" />
    <Rectangle Canvas.Left="110" Canvas.Top="110" Height="200" Width="200" Stroke="Black"
               StrokeThickness="10" Fill="Green" />
</Canvas>
```



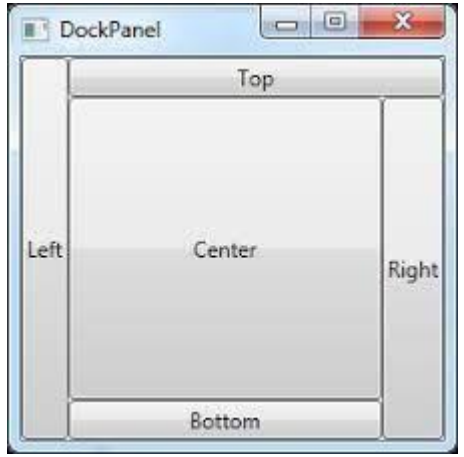
Layout - DockPanel

DockPanel defines an area to arrange child elements relative to each other

- horizontally or
- vertically.

With DockPanel you can easily dock child elements using the Dock property to

- top,
- bottom,
- right,
- left and
- center.




```
<DockPanel>
    <Button DockPanel.Dock="Left">Left</Button>
    <Button DockPanel.Dock="Top">Top</Button>
    <Button DockPanel.Dock="Right">Right</Button>
    <Button DockPanel.Dock="Bottom">Bottom</Button>
    <TextBox>Center</TextBox>
</DockPanel>
```

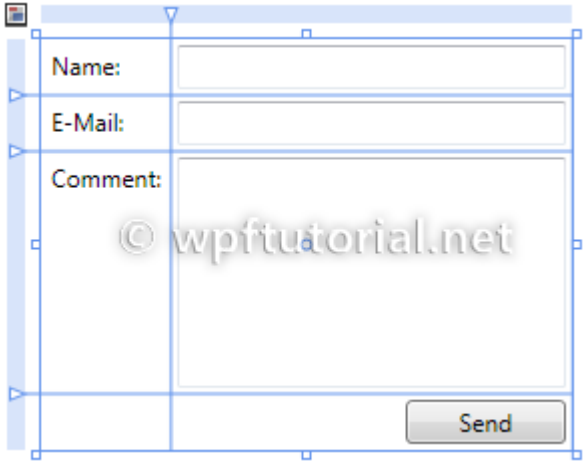
Layout - Grid

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
        <RowDefinition Height="28" />
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="200" />
    </Grid.ColumnDefinitions>

    <Label Grid.Row="0" Grid.Column="0" Content="Name:"/>
    <Label Grid.Row="1" Grid.Column="0" Content="E-Mail:"/>
    <Label Grid.Row="2" Grid.Column="0" Content="Comment:"/>

    <TextBox Grid.Row="0" Grid.Column="1" Margin="3" />
    <TextBox Grid.Row="1" Grid.Column="1" Margin="3" />
    <TextBox Grid.Row="2" Grid.Column="1" Margin="3" />

    <Button Grid.Row="3" Grid.Column="1" HorizontalAlignment="Right"
            MinWidth="80" Margin="3" Content="Send" />
</Grid>
```

The grid is a layout panel that arranges its child controls in a tabular structure of rows and columns.

Its functionality is similar to the HTML table but more flexible.

A cell can contain multiple controls, they can span over multiple cells and even overlap themselves.

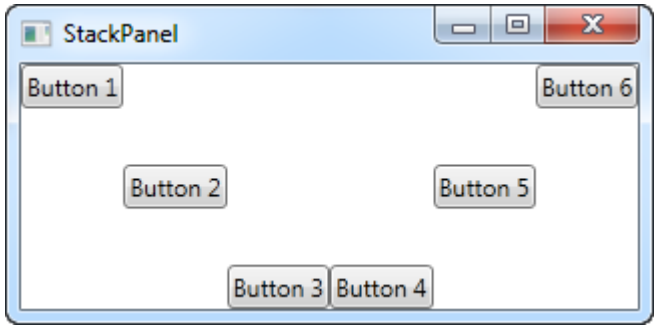


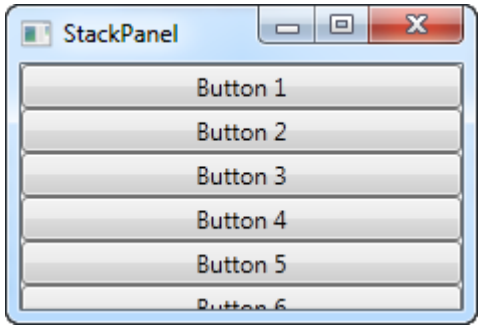
Layout - StackPanel

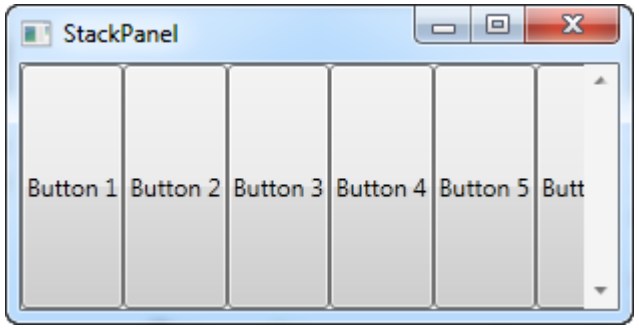
```
<StackPanel Orientation="Horizontal">
    <Button VerticalAlignment="Top">Button 1</Button>
    <Button VerticalAlignment="Center">Button 2</Button>
    <Button VerticalAlignment="Bottom">Button 3</Button>
    <Button VerticalAlignment="Bottom">Button 4</Button>
    <Button VerticalAlignment="Center">Button 5</Button>
    <Button VerticalAlignment="Top">Button 6</Button>
</StackPanel>
```

Arrange control either vertically or horizontally.

At the time of stack layout creation, you have to set the property Orientation either as Vertical or as Horizontal.

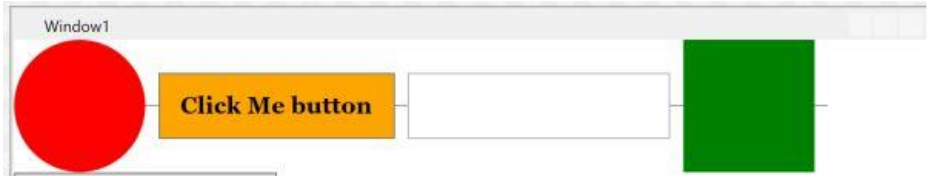






Layout - WrapPanel

```
<WrapPanel>
    <Ellipse Width="100" Height="100" Fill="Red" />
    <Separator Width="10" />
    <Button Background="Orange" Width="180" Height="50“
            FontFamily="Georgia" FontSize="18“
            FontWeight="Bold" Name="ClickMeButton">
            Click Me button
    </Button>
    <Separator Width="10" />
    <TextBox Height="50" Width="200" Name="Button1" />
    <Rectangle Width="100" Height="100" Fill="Green" />
</WrapPanel>
```



WPF WrapPanel control is a panel that positions child elements in sequential position from left to right by default.

If child elements that are stacked don’t fit in the row or column they are in, the remaining elements will wrap around in the same sequence.

See example of horizontal wrap panel

Namespaces in XAML

At the beginning of every XAML file you need to include two namespaces.

http://schemas.microsoft.com/winfx/2006/xaml/presentation.

- It is mapped to all wpf controls in System.Windows.Controls.

http://schemas.microsoft.com/winfx/2006/xaml

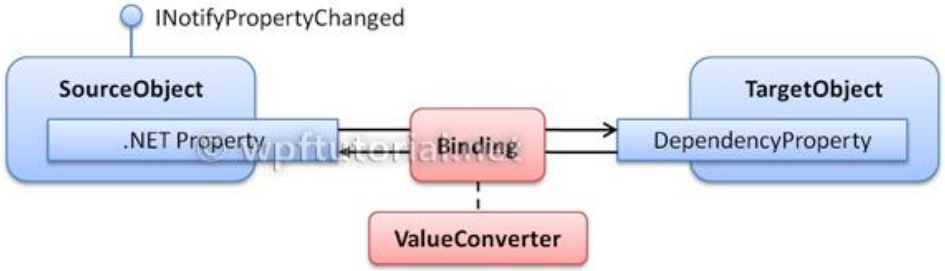
- it is mapped to System.Windows.Markup that defines the XAML keywords.

You can also directly include a CLR namespace in XAML by using the clr-namespace: prefix.

Example

```
<Window x:Class="Week13Day3Test.MainWindow“
        xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
        xmlns:local="clr-namespace:Week13Day3Test">
```

Data binding overview



Data Binding is a simple and powerful way to auto-update data between the business model and the user interface.

Every time when the data of your business model changes, it automatically reflects the updates to the user interface and vice versa.

This is the preferred method in WPF to bring data to the user interface.

Databinding can be unidirectional (source -> target or target <- source), or bidirectional (source <-> target).

Like in WinFoms, you set properties on a control manually like populate a ListBox by adding items to it from a loop

But the cleanest and purest WPF way is to add a binding between the source object and the destination UI element.

Data binding Hello World

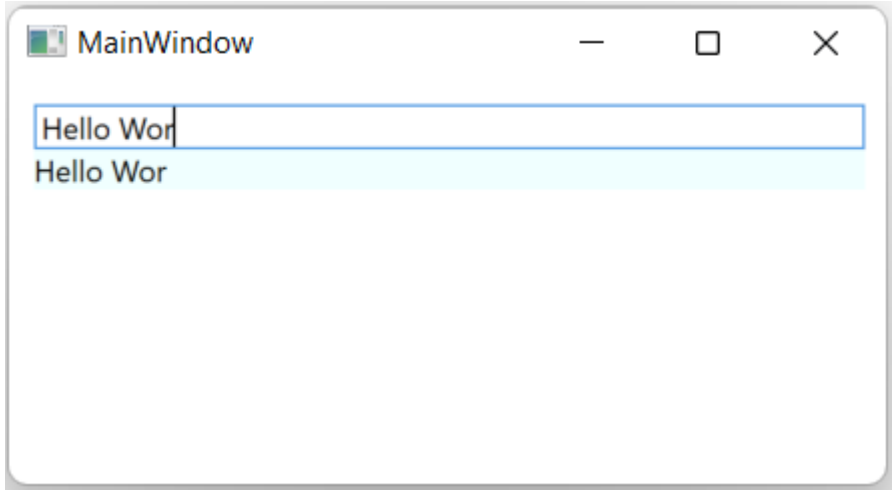
This example shows how we bind the value of the TextBlock to match the Text property of the TextBox.

The TextBlock is automatically updated when you enter text into the TextBox.

In WinForms, this would require us to listen to the KeyPressed event on the TextBox and then update the TextBlock each time the text changes

In WPF, with data binding, the connection between the two controls can be established automatically just by specifying it in xaml.

```
<StackPanel Margin="10">
   <TextBox Name="txtValue" />
   <TextBlock
      Text="{Binding Path=Text, ElementName=txtValue}" />
</StackPanel>
```



The syntax of a Binding

The binding magic happens between the curly braces

In XAML the curly braces contain a Markup Extension.

Binding is a type of Markup Extension, which allows us to describe the binding relationship for the Text property. In its most simple form, a binding can look like this:

{Binding Path=Text, ElementName=txtValue}

Using DataContext

During binding you can specify ElementName on every control, when you want to bind to another element

- `<TextBlock Text="{Binding Path=Text, ElementName=txtValue}" />`

However, the DataContext property is the default source of your bindings when you want to bind to an object in your Model

- `<TextBlock Text="{Binding Path=Name}" />`
- Or
- `<TextBlock Text="{Binding Name}" />`
- In Code-Behind
- `this.DataContext = <object>;`