**WPF (Window Presentation Foundation)**

WPF is basically used for to develop an interactive desktop application.

For Developing a WPF Application we can use a XAML file, XAML is an XML based declarative markup language for specifying and setting the characteristics of classes. In other words, XAML is a language used by WPF,

**Few Things to Know Before Start First Application.**

### **What is meant by Thread Affinity?**

When WPF application starts, it actually creates two threads automatically. One is Rendering Thread, which is hidden from the programmer, so you cannot use the rendering thread directly from your program; while the other is Dispatcher Thread, which actually holds all the UI elements. So in other words, you might say Dispatcher is actually the UI thread which ties all the elements created within the WPF application. Conversely, WPF requires all the UI elements to be tied with Dispatcher thread, this is called Thread Affinity. Thus you cannot change any element created on Dispatcher thread from any other threads.

### **Why Dependency Object is used?**

Every WPF control is derived from DependencyObject. DependencyObject is a class that supports DependencyProperty, a property system that is newly built in WPF. Every object is derived from DependencyObject and hence it can associate itself in various inbuilt features of WPF like EventTriggers, PropertyBindings, Animations, etc.

Every DependencyObject actually has an Observer or a List and declares 3 methods called ClearValue, SetValue and GetValue which are used to add/edit/remove those properties. Thus the DependencyProperty will only create itself when you use SetValue to store something. Thus, it is resource saving as well.

**• Building Your First WPF Application**

<Window x:Class="FirstWindowsApplication.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

x:Name="Window1"

Title="Window1" Height="300" Width="300">

<Grid>

</Grid>

</Window>

## Border is the primary building block of any application in WPF, borders generally play a very important role in creating a better look and feel for the application. In this application, you will see how you can use Borders and most of the properties with ease.

<Border Width="50" Height="50" x:Name="brdElement">

<Border.Background>

<SolidColorBrush Color="Bisque"></SolidColorBrush>

</Border.Background>

<Border.Effect>

<DropShadowEffect BlurRadius="10" Color="Red" Direction="235" Opacity=".5"

RenderingBias="Quality" ShadowDepth="10" />

</Border.Effect>

</Border>

1. **Color**: Defines the Color of the Shadow.
2. **Opacity**: Fades out the Color. You can see the Red color is faded out to .5; Opacity ranges between 0 - 1.
3. **BlurRadius**: It defines the extent of shadow radius. Thus if you increase the size of BlurRadius, it will increase the Shadow.
4. **Direction**: It is the Light Direction in degrees. 235 degree implies where the shadow will focus, thus you can see 360 -235 is the angle where light is placed. Value ranges from 0 to 360.
5. **ShadowDepth**: It defines the depth of the Shadow. It means, how much the object is raised from the Shadow. If you increase the value of ShadowDepth, you will see, the being raised.

# Introduction to WPF panels

* **Canvas**

It allows you to assign specific coordinates to each of the child controls, giving you total control of the layout. This is not very flexible though, because you have to manually move the child controls around and make sure that they align the way you want them to. Use it (only) when you want complete control of the child control positions.

<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>

* **WrapPanel**

Use it when you want a vertical or horizontal list controls that automatically wraps when there's no more space.

<WrapPanel>

<Button>Test button 1</Button>

<Button>Test button 2</Button>

<Button>Test button 3</Button>

<Button Height="40">Test button 4</Button>

<Button>Test button 5</Button>

<Button>Test button 6</Button>

</WrapPanel>

* **StackPanel**

The StackPanel is very similar to the WrapPanel, but with at least one important difference: The StackPanel doesn't wrap the content. Instead it stretches it content in one direction, allowing you to stack item after item on top of each other.

<StackPanel Margin="0,48.803,0,20.141">

<Button>Button 1</Button>

<Button>Button 2</Button>

<Button>Button 3</Button>

<Button>Button 4</Button>

<Button>Button 5</Button>

</StackPanel>

<StackPanel Orientation="Horizontal" Margin="71.02,36.083,129.333,3.18">

<Button>Button 1</Button>

<Button>Button 2</Button>

<Button>Button 3</Button>

<Button>Button 4</Button>

<Button>Button 5</Button>

</StackPanel>