WPF Interview questions with answers

By

www.questpond.com

Call for WPF training 91- 022- 66752917

What is WPF?

WPF (Windows Presentation foundation) is a graphical subsystem for displaying user interfaces, documents, images, movies etc in windows application.

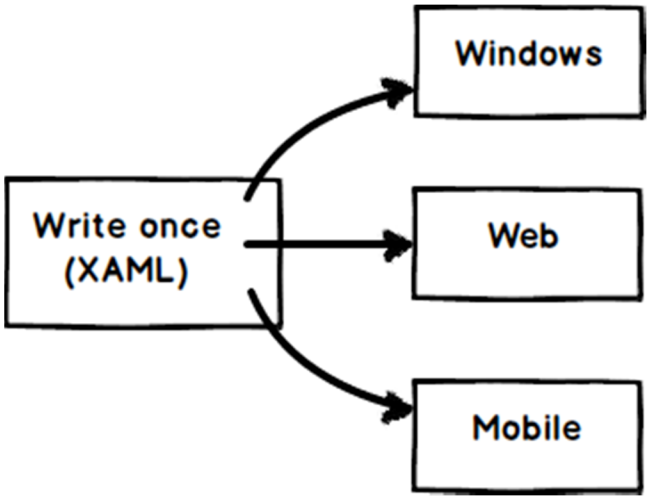
What is the need of WPF when we had windows forms?

```
Remember: - ABCDEFG
```

A - Anywhere execution (Windows or Web)
B - Bindings (less coding)
C - Common look and feel (resource and styles)
D - Declarative programming (XAML)
E - Expression blend animation (Animation ease)
F - Fast execution (Hardware acceleration)
G - Graphic hardware independent (resolution independent)

What is XAML in WPF and why do we need it?

XAML is a XML file which represents your WPF UI. The whole point of creating the UI representation in XML was write once and run it anywhere. So the same XAML UI can be rendered as windows application with WPF and the same UI can be displayed on the browser using WPF browser or Silverlight application.



What is xmlns in XAML file ?

"xmlns" stands for XML namespaces. It helps us to avoid name conflicts and confusion in XML documents. For example consider the below two XML which have table elements, one table is a HTML table and the other represents a restaurant table. Now if both these elements come in a single XML document there would name conflicts and confusion.

```
<table>
<tr><td>Row1</td></tr>
<tr><td>Row2</td></tr>
</table>
```

```
<table>
<cloth>red</cloth>
<serve>Tea</serve>
</table>
```

So to avoid the same we can use XML namespaces. You can see in the below XML we have qualified the HTML table with "<h:table>" and the restaurant table element qualified with "<r:table>".

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
<tr><td>Row1</td></tr>
<tr><td>Row2</td></tr>
</h:table>
```

```
<r:table xmlns:h="http://www.questpond.com/restaurant/table/def">
<cloth>red</cloth>
<serve>Tea</serve>
</r:table>
```

What is the difference between xmlns and xmlns:x in WPF ?

Bothe namespaces helps to define / resolved XAML UI elements.

The first namespace is the default namespace and helps to resolve overall WPF elements.

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

The second namespace is prefixed by “x:” and helps to resolve XAML language definition.

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

For instance for the below XAML snippet , we have two things one is the “StackPanel” and the other is “x:name”. “StackPanel” is resolved by the default namespace and the “x:name” is resolved by using “xmlns:x” namespace.

```
<StackPanel x:Name="myStack" />
```

Provide some instances where you have “xmlns:x” namespace in XAML ?

There are two common scenarios where we use “xmlns:x” namespace :-

To define behind code for the XAML file using “x:class” attribute.

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="MyNamespace.MyCanvasCodeInline"
>
```

Second to provide name to an element.

```
<StackPanel x:Name="myStack" />
```

When should we use “x:name” vs “name” ?

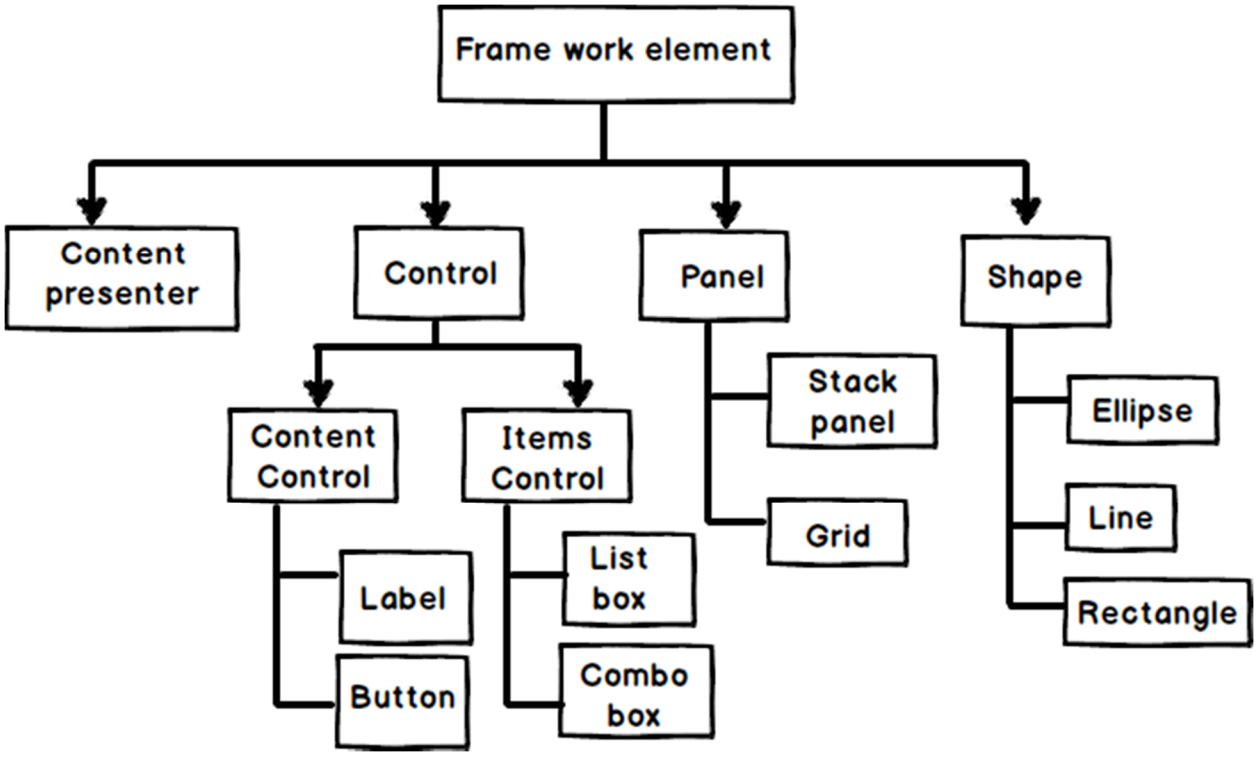
There is no difference between “x:name” and “name” , “name” is short hand of “x:name”. But in classes where you do not find “name” property (this is a rare situations) we need to use “x:name” property.

What are the different kinds of controls in WPF?

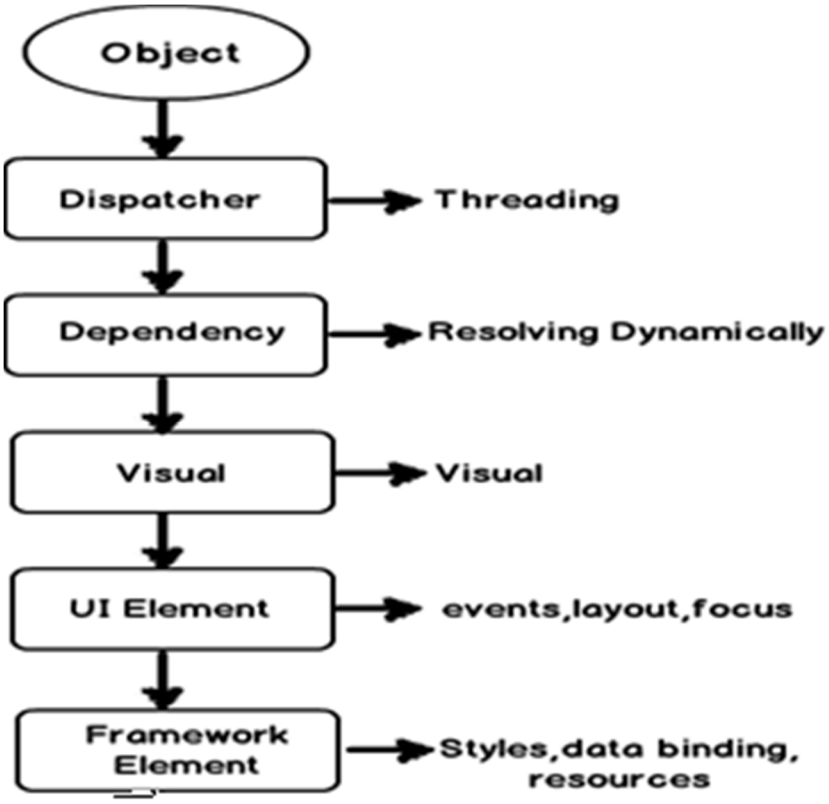
WPF controls can be categorized in to four categories:-

- **Control**: - This is the basic control with which you will work most of time. For example textbox, buttons etc. Now controls which are standalone control like button , text box , labels etc are termed as content control. Now there are other controls which can hold other controls, for instance itemscontrols. Itemscontrol can have multiple textbox controls, label controls etc.
- **Shape**: - These controls help us to create simple graphic controls like Ellipse, line, rectangle etc.
- **Panel**: - These controls help to align and position the controls. For instance grid helps us to align in a table manner, stack panel helps for horizontal and vertical alignment.
- **Content presenter**: - This control helps to place any XAML content inside it. Used when we want to add dynamic controls on a WPF screen.

All the above four types of WPF controls finally inherit from the frameworkelement class of WPF.



Can you explain the complete WPF object hierarchy?



Object: - As WPF is created using .NET so the first class from which WPF UI classes inherits is the .NET object class.

Dispatcher: - This class ensures that all WPF UI objects can be accessed directly only by the thread who own him. Other threads who do not own him have to go via the dispatcher object.

Dependency: - WPF UI elements are represented by using XAML which is XML format. At any given moment of time a WPF element is surrounded by other WPF elements and the surrounded elements can influence this element and this is possible because of this dependency class. For example if a textbox surrounded by a panel, its very much possible that the panel background color can be inherited by the textbox.

Visual: - This is the class which helps WPF UI to have their visual representation.

UI Element: - This class helps to implement features like events, input, layouting etc.

Framework element: - This class supports for templating , styles , binding , resources etc.

And finally all WPF controls textbox , button , grids and whatever you can think about from the WPF tool box inherits from the framework element class.

Does that mean WPF has replaced DirectX?

No, WPF does not replace DirectX. DirectX will still be still needed to make cutting edge games. The video performance of directX is still many times higher than WPF API. So when it comes to game development the preference will be always DirectX and not WPF. WPF is not an optimum solution to make games, oh yes you can make a TIC TAC TOE game but not high action animation games.

One point to remember WPF is a replacement for windows form and not directX.

So is XAML meant only for WPF ?

No,XAML is not meant only for WPF.XAML is a XML-based language and it had various variants.

WPF XAML is used to describe WPF content, such as WPF objects, controls and documents. In WPF XAML we also have XPS XAML which defines an XML representation of electronic documents.

Silverlight XAML is a subset of WPF XAML meant for Silverlight applications. Silverlight is a cross-platform browser plug-in which helps us to create rich web content with 2-dimensional graphics, animation, and audio and video.

WWF XAML helps us to describe Windows Workflow Foundation content. WWF engine then uses this XAML and invokes workflow accordingly.

Can you explain the overall architecture of WPF?

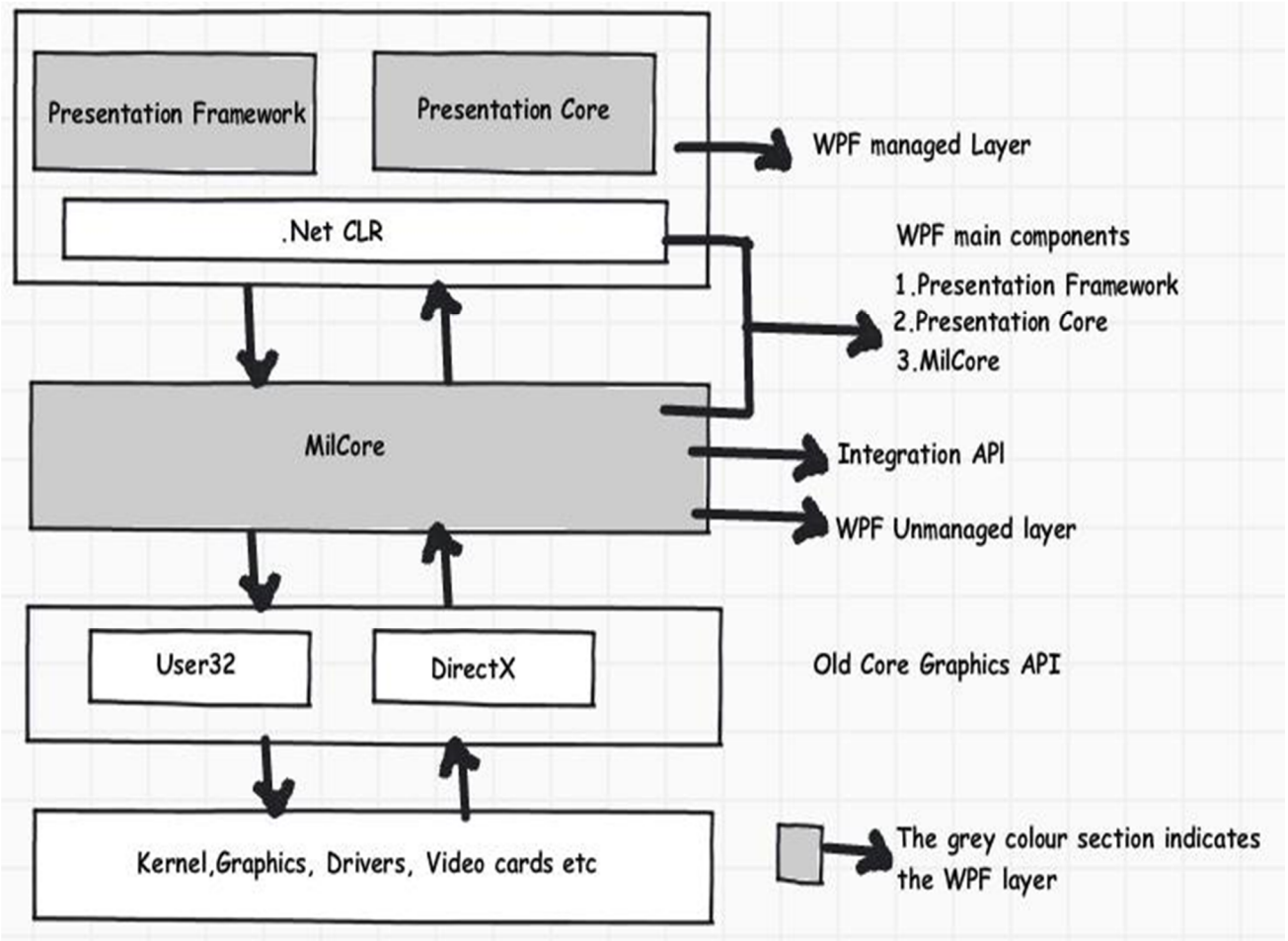


Figure 8.2: - Architecture of WPF

Above figure shows the overall architecture of WPF. It has three major sections presentation core, presentation framework and milcore. In the same diagram we have shown how other section like direct and operating system interact with the system. So let's go section by section to understand how every section works.

User32:- It decides which goes where on the screen.

DirectX: - As said previously WPF uses directX internally. DirectX talks with drivers and renders the content.

Milcore: - Mil stands for media integration library. This section is a unmanaged code because it acts like a bridge between WPF managed and DirectX / User32 unmanaged API.

Presentation core ;- This is a low level API exposed by WPF providing features for 2D , 3D , geometry etc.

Presentation framework:- This section has high level features like application controls , layouts . Content etc which helps you to build up your application.

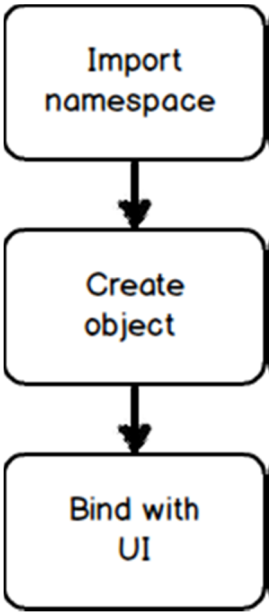
What is App.xaml in WPF project?

App.xaml is the start up file or a boot strapper file which triggers your first XAML page from your WPF project.

What are resources in WPF ?

Resources are objects referred in WPF XAML. In C# code when we create an object we do the following three steps :-

```
using CustomerNameSpace; // import the namespace.

Customer obj = new Customer(); // Create object of the class

Textbox1.text = obj.CustomerCode; // Bind the object with UI elements
```



So even in WPF XAML to define resources which are nothing but objects we need to the above 3 steps :-

- Import namespace where the class resides: - To define namespace we need to use the "xmlns" attribute as shown in the below XAML code.

```
<Window x:Class="LearnWpfResources.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:custns="clr-namespace:LearnWpfResources"
        Title="MainWindow" Height="350" Width="525">
```

- Create object of the class :- To create an object of the class in XAML we need to create a resource by using the resource tag as the below code. You can the object name is ‘custobj”.

```
<Window.Resources>
        <custns:Customer x:Key="custobj"/>
  </Window.Resources>
```

The above code you can map to something like this in C#

```
Customer custobj = new Customer();
```

- Bind the object with UI objects :- Once the object is created we can then bind them using bindings like one way , two way as explained in “Explain one way, two way, one time and one way to source?” question explained above.

```
<TextBox Text="{Binding CustomerCode, Mode=TwoWay, Source={StaticResource
custobj}}" />
```

Explain the difference between static and dynamic resource?

Resources can be referred statically or dynamically. Static referred resources evaluate the resource only once and after that if the resources change those changes are not reflected in the binding. While dynamic referred resources are evaluated every time the resource is needed.

Consider the below “SolidColorBrush” resource which is set to “LightBlue” color.

```
<Window.Resources>
        <SolidColorBrush Color="LightBlue" x:Key="buttonBackground" />
 </Window.Resources>
```

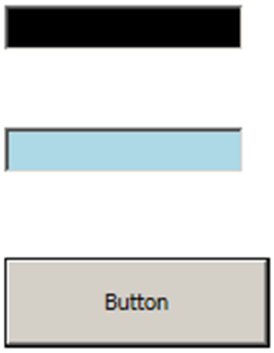
The above resource is binded with the below two textboxes statically and dynamically respectively.

```
<TextBox Background="{DynamicResource buttonBackground}" />
<TextBox Background="{StaticResource buttonBackground}"/>
```

Now if we modify the resource , you see the first text box changes the background and the other textbox color stays as it is.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
          Resources["buttonBackground"] = Brushes.Black;
}
```

Below is the output of the same.

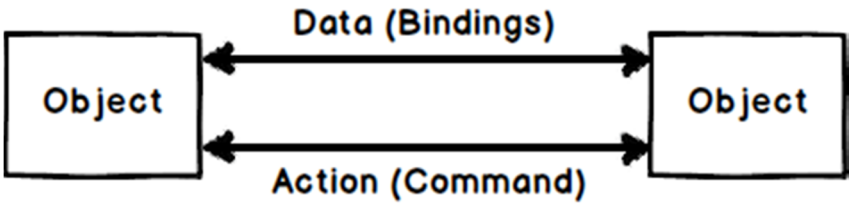


When should we use static resource over dynamic resource ?

Dynamic resources reduce application performance because they are evaluated every time the resource is needed. So the best practice is use Static resource until there is a specific reason to use dynamic resource. If you want resource to be evaluated again and again then only use dynamic resource.

Explain the need of binding and commands?

WPF Binding's helps to send / receive data between WPF objects while command helps to send and receive actions. The object that emits data or action is termed as **source** and the object who wants to receive data or action is termed as **target**.



Explain one way, two way, one time and one way to source?

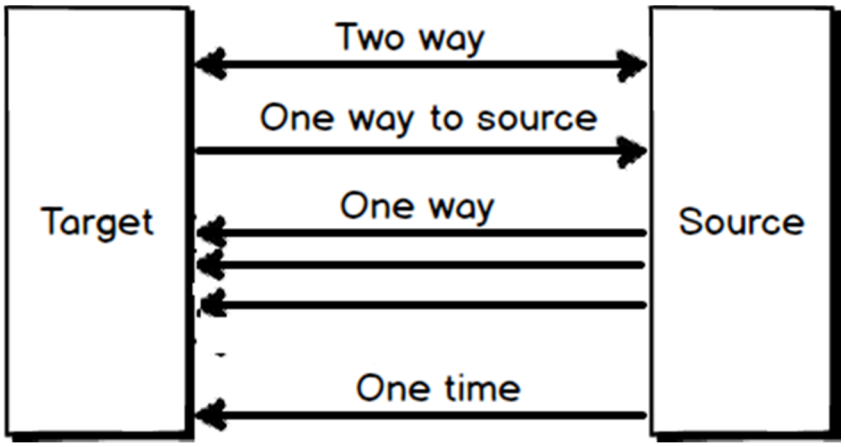
All the above 4 things define how data will flow between target and source objects when WPF binding is applied.

Two way: - Data can flow from both source to target and from target to source.

One way: - Data flows only from source to target.

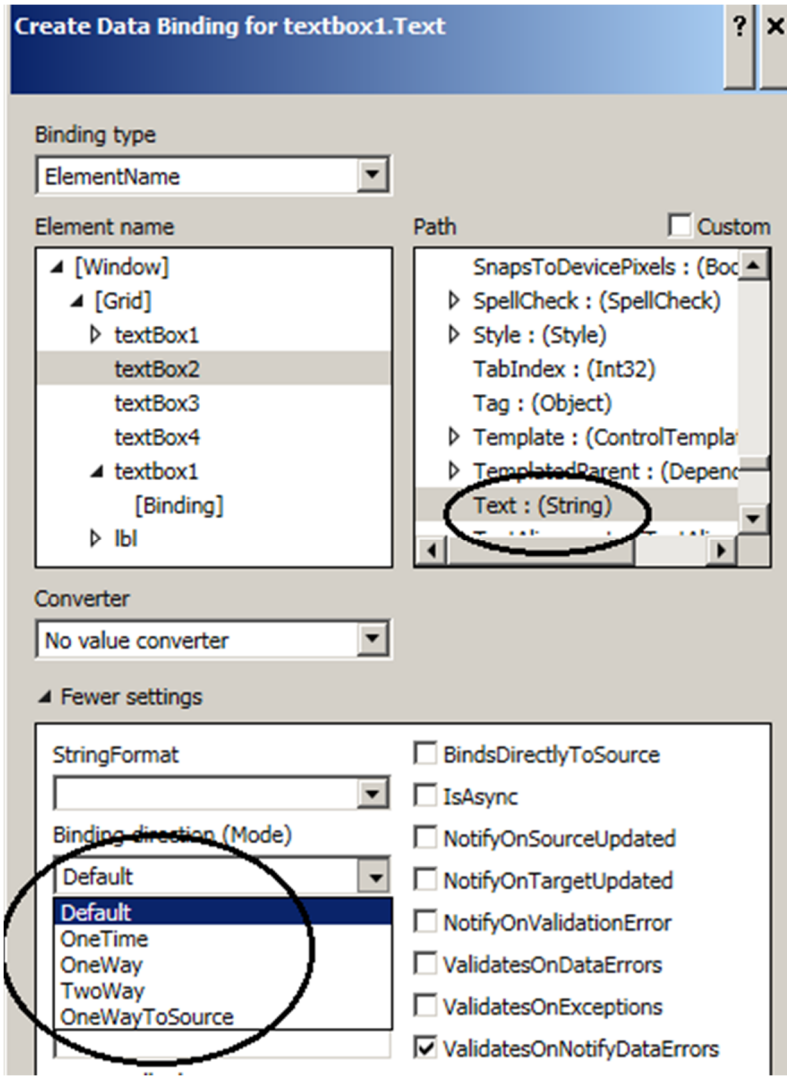
One way to source: - Data flows only from target to source.

One time: - Data flows only for the first time from source to target and after that no communication happens.



Below is an easy tabular representation to memorize the same.

	Data flow	
Bindings	**Target to Source**	**Source to target**
Two way	Yes	Yes
One way to source	Yes	No
One way	No	Yes
One time	No	For first Yes



Can you explain WPF command with an example?

When end users interact with application they send actions like button click, right click , control + c, control + v etc. A command class in WPF wraps these end user actions in to a class so that they can be reused again and again.

WPF Command class idea is an implementation of command pattern from gang of four design pattern.

To create a command class we need to implement the “ICommand” interface. For example below is a simple command class which increments a counter class by calling “Increment” method.

```
public class IncrementCounter : System.Windows.Input.ICommand
    {
        private clsCounter obj;
        public IncrementCounter(clsCounter o)
        {
            obj = o;
        }
        public bool CanExecute(object parameter)
        {
            return true;
        }

        public event EventHandler CanExecuteChanged;

        public void Execute(object parameter)
        {
            obj.Increment();
        }
    }
```

The command class needs two implement two methods as shown in the above code:-

What to Execute (Execute) – Command class is all about wrapping actions of end users so that we can reuse them. At the end of the day Action invokes methods. Mean for instance a "btnmaths_click" action will invoke "Add" method of a class. The first thing we need to specify in the command class is which method you want to execute. In this case we want to call the "Increment" method of "clsCounter" class.

When to execute (CanExecute) – The second thing we need to specify is when the command can execute, means validations. This validation logic is specified in the "CanExecute" function. If the validation returns true then "Execute" fires or else the action has no effect.

You can see the code of "CanExecute" and "Execute" in the above code snippet.
Once you have created the "Command" class you can bind this with the button using the "Command" property as shown in the below XAML code."CounterObj" is a resource object.

```
<Button Command="{Binding IncrementClick, Source={StaticResource
Counterobj}}"/>
```

So now that the action is wrapped in to command we do not need to write method invocation code again and again in behind code, we just need to bind the command object with the WPF UI controls wherever necessary.

How does "UpdateSourceTrigger" affect bindings?

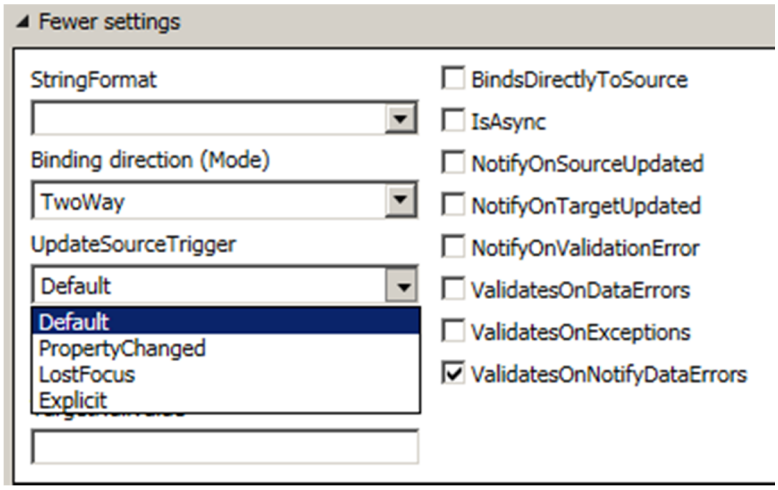
"UpdateSourceTrigger" decides when the data should get updated between WPF objects that are binded. In other word should data get updated in lost focus event, in data change event etc.

There are four modes by which "UpdateSourceTrigger" can be defined:-

- Default: - If it's a text property then data is updated during lost focus and for normal properties data updates in property change event.
- PropertyChanged: - In this setting data is updated as soon as the value is changed.
- LostFocus: - In this setting data is updated as soon as lost focus event occurs.
- Explicit: - In this setting the data is updated manually. In other words to update data between two WPF object you need to call the below code.

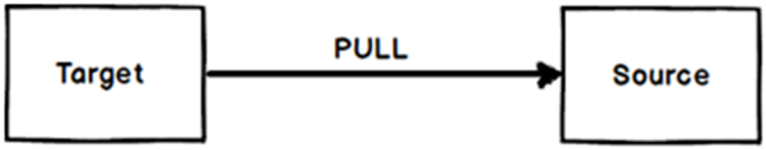
```
BindingExpression binding =txt1.GetBindingExpression(TextBox.TextProperty);
binding.UpdateSource();
```

To set "UpdateSourceTrigger" we need to use the "UpdateSourceTrigger" value which can be found in the bindings properties as shown in the below figure.

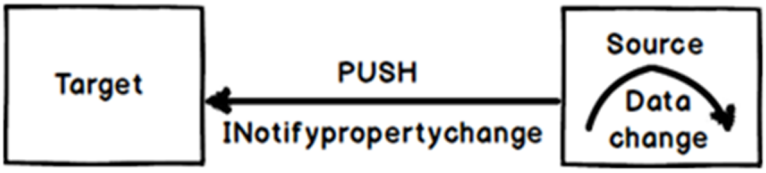


Explain the need of “INotifyPropertyChanged” interface?

When we bind two WPF objects the target data is updated depending on the “UpdateSourceTrigger” events. Please refer the previous question for “UpdateSourceTrigger” basics.
The “UpdateSourceTrigger” has events like lostfocus , property change etc. In other words when lostfocus or property change event happen on the target it makes a PULL to the source to get the latest data.



So it’s very much possible that the WPF source data has changed and because the WPF target “UpdateSourceTrigger” event did not fire he did not make a pull and the data of the source is not in synch with the target. This is where “INotifyPropertyChanged” interface comes to use.



Below is a simple “clsCounter” class which has a “Counter” property and this property is incremented by “Increment” method.

Now if we bind WPF label or textbox to the “Counter” property and call the “Increment” method the new “Counter” value will not be propagated to the target. Because invoking a method does not trigger any “UpdateSourceTrigger” event.

So after calling the “Increment” method the “Counter” value of the source and the target are out of synch.

So create a push event from the source you need to first implement “INotifyPropertyChanged” interface as shown in the below figure. Now when someone calls the “Increment” method you can raise an event saying that the “Counter” property has changed by calling “PropertyChanged” function as shown in the below code.

In simple words the source sends a notification to the target WPF object that data has changed in the source and he should refresh himself with the fresh data.

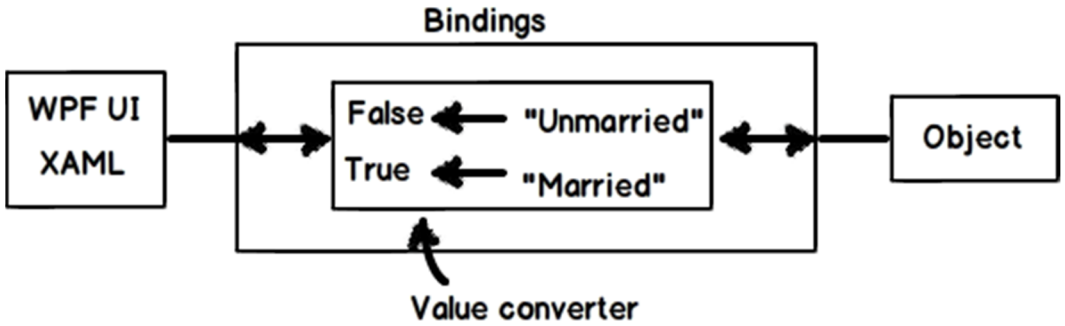
```
PropertyChanged(this, new PropertyChangedEventArgs("Counter"));
```

Below is full “clsCounter” class code with “INotifyPropertyChanged” implemented.

```
public class clsCounter : INotifyPropertyChanged
{

        private int _Counter=0;

        public int Counter
        {
            get { return _Counter; }
        }

        public void Increment()
        {
            _Counter++;
            PropertyChanged(this, new PropertyChangedEventArgs("Counter"));
        }


         public event PropertyChangedEventHandler PropertyChanged;
   }
```

What are value converters in WPF?

Binding is one of the big features in WPF which helps us to facilitate data flow between WPF UI and Object. But when data flows from source to UI or vice-versa using these bindings we need to convert data from one format to other format. For instance let's say we have a "Person" object with a married string property.



Let's assume that this "married" property is binded with a check box. So if the text is "Married" it should return true so that the option check box look's checked. If the text is "Unmarried" it should return false so that the option check box looks unchecked.

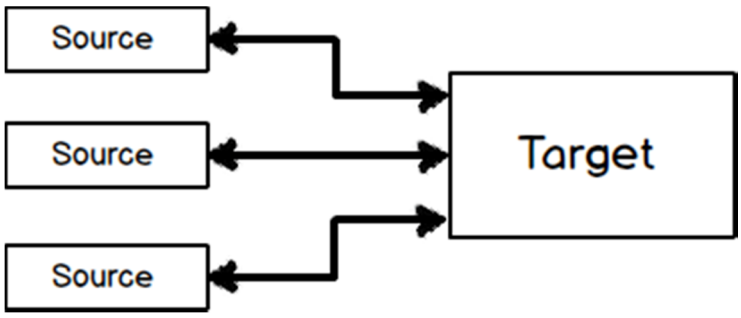
In simple words we need to do data transformation. This is possible by using “Value Converters”. In order to implement value converters we need to inherit from “IValueConverted” interface and implement two methods “Convert” and “ConvertBack”.

```
public class MaritalConverter : IValueConverter
{
        public object Convert(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
        {
            string married = (string)value;
            if (married == "Married")
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        public object ConvertBack(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
        {
            bool married = (bool)value;
            if (married)
            {
                return "Married";
            }
            else
            {
                return "UnMarried";
            }
        }
    }
```

You can see in the “Convert” function we have written logic to transform “Married” to true and from “Unmarried” to false. In the “Convertback” function we have implemented the reverse logic.

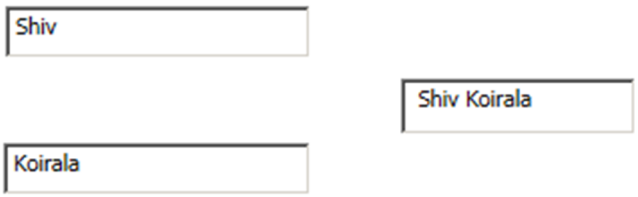
Explain multi binding and multivalue converters in WPF?

“MultiBinding” helps you bind multiple sources to a target while multi-converters acts like bridge if the source and targets have different data formats or need some conversion.



For example let’s say you have two textboxes which has “FirstName” and “LastName”. You want that as soon as users type on these two textboxes, the other text box should get updated with “FirstName LastName” (As shown in the below figure).

Also vice versa if we type in the third text box “FirstName LastName” it should display “FirstName” and “LastName’ in the other two textboxes respectively.



So the first thing is to create a multivalue converter class which will join “FirstName” and “LastName” in to source and split “FirstName LastName” back to target. For the same we need to implement “IMultiValueConverter” and implement “Convert” and “ConvertBack” methods.
“Convert” helps to do conversion from “Target” to “Source” and “ConvertBack” helps to convert from “Source” to “Target”.

```
public class NameMultiConverter : IMultiValueConverter
{
public object Convert(object[] values, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
// Conversion logic from Target to Source
}
```

```
}
public object[] ConvertBack(object value, Type[] targetTypes, object
parameter, System.Globalization.CultureInfo culture)
{
// Conversion logic from Source to Target.
}
}
```

Below is the "Convert" (Target to Source) implementation where we get the "textbox" value in a array and we have concatenated the array values and returned the same in a single string. This single string will be displayed in the third textbox.

```
public object Convert(object[] values, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
        {
            string x = "";
            foreach (object y in values)
            {

                x = x + " " + y.ToString();
            }
            return x;
        }
```

"ConvertBack" helps to implement conversion logic from "Source" to "Target". So when someone types "Shiv Koirala" in the third text box we would like to split "Shiv" in one string and "Koirala" in other string.
So you can see in the below code we have used the "Split" function to create a string array. So the first array index (array[0]) will go to the first textbox and the second array index (array[1]) will go to the second text box.

```
public object[] ConvertBack(object value, Type[] targetTypes, object
parameter, System.Globalization.CultureInfo culture)
        {
            string str = (string)value;
            string[] val = str.Split(' ');



            return val;
```

```
        }
```

Now once we are done with the converter next thing is to apply multi-binding. Below is the XAML code where we have the three text boxes “txtFirstName” , “txtLastName” and “txtFirstAndLast”.

You can see how the “txtFirstAndLast” textbox “Text” is binded using multibinding element which binds to “txtFirstName” and “txtLastName” textbox . So now if you make changes to multiple target the single source gets updated and if you update the source multiple target gets updates.

```
<TextBox x:Name="txtFirstName" HorizontalAlignment="Left" Height="26"
Margin="42,67,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top"
Width="152"/>

<TextBox x:Name="txtLastName" HorizontalAlignment="Left" Height="26"
Margin="41,135,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top"
Width="153"/>

<TextBox x:Name="txtFirstAndLast"  Height="28" HorizontalAlignment="Left"
Margin="239,103,0,0" Name="label1" VerticalAlignment="Top" Width="117">
<TextBox.Text>
<MultiBinding Converter="{StaticResource NameMultiConverter}">
                    <Binding ElementName="txtFirstName" Path="Text" />
                    <Binding ElementName="txtLastName" Path="Text" />
                </MultiBinding>
</TextBox.Text>
</TextBox>
```

Explain WPF relative binding / relative resource?

When we define bindings we need at least two elements target and source. But many times rather than defining binding between two elements we would like to define binding with reference to the current element i.e. RELATIVELY.

For instance let’s say we have a WPF border and we would like height and width of the broder to be same. So for this scenario the target and source are the same, the WPF border itself. So we can define the binding using “RelativeSource” as shown in the below code. You can see it uses “Self” binding mode to bind the element to itself.

```
<Border BorderBrush="Black" BorderThickness="1" Height="139"
Width="{Binding Height,  RelativeSource={RelativeSource Self}}"/>
```

What are the different ways of binding using relative source?

There are four ways of binding relatively in WPF :-

- Self
- Ancestor
- Previousdata
- Templated parent

Can you explain self relative source binding in WPF?

This relative binding helps to bind to one property of an element to the other property of the same element. For example in the below XAML the border width is binded to height of the same border element.

```
<Border BorderBrush="Black" BorderThickness="1" Height="139"
Width="{Binding Height,  RelativeSource={RelativeSource Self}}"/>
```

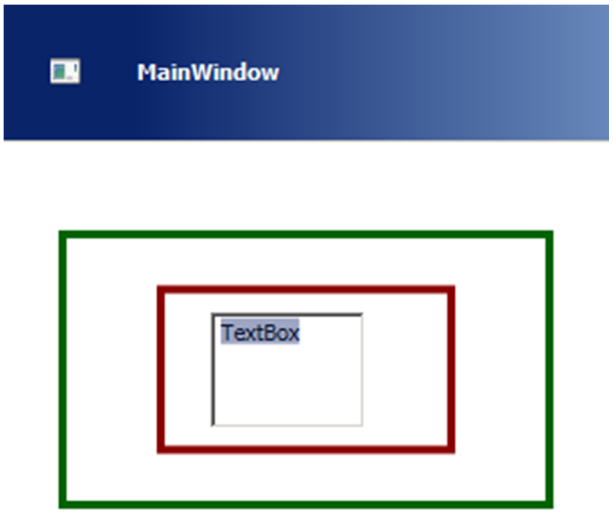
Explain Ancestor relative source binding in WPF?

This relative binding helps to bind properties to the parent element properties. For example in the below XAML code we have a textbox which has two border's as a parent. One border is having dark green and the other border is having dark red color as the border color.

The dark green color border is the parent element followed by dark red and the text box the child element at the end of the hierarchy.

```
<Border BorderBrush="DarkGreen">
<Border BorderBrush="DarkRed”>
<TextBox />
</Border>
</Border>
```

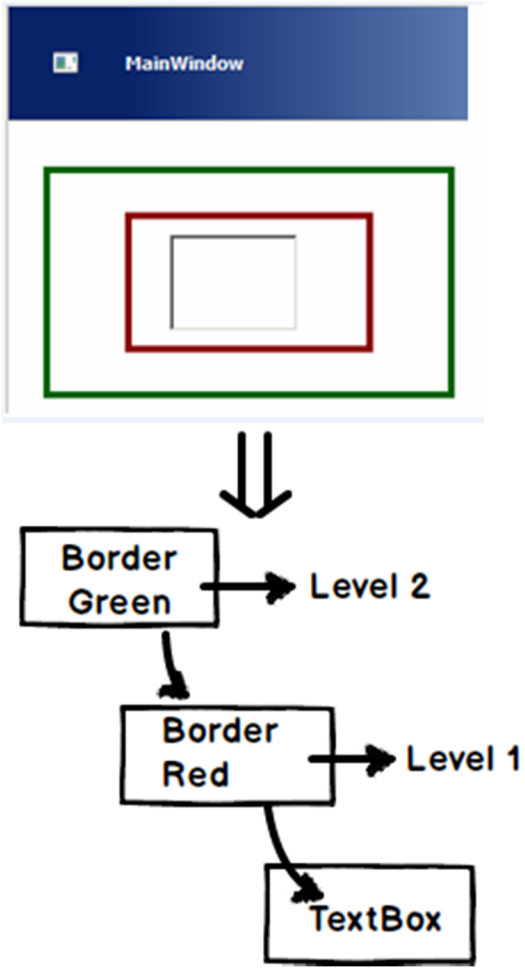
Below is how the WPF application looks like when it runs.



Now we want the background color of the text box to be binded to one of the parent border colors. To achieve the same we can use ancestor relative binding.

Below are some important properties in Ancestor type binding we need to know before we writing the binding code.

Property	**Description**
AncestorType	Which type of parent element is it?. Is it a border element , grid element etc.
AncestorLevel	An element can have multiple parents. For example in the above XAML we have two parents one with red color and the other with green. So this property specifies which level of parent element we want to refer. Below is the figure which depicts the levels. The red color border is level 1 and the green color border is level 2. So the nearest element becomes the first level and so on.
Binding	This specifies which property we want to bind to. For the current example we want to bind to border's brush color.



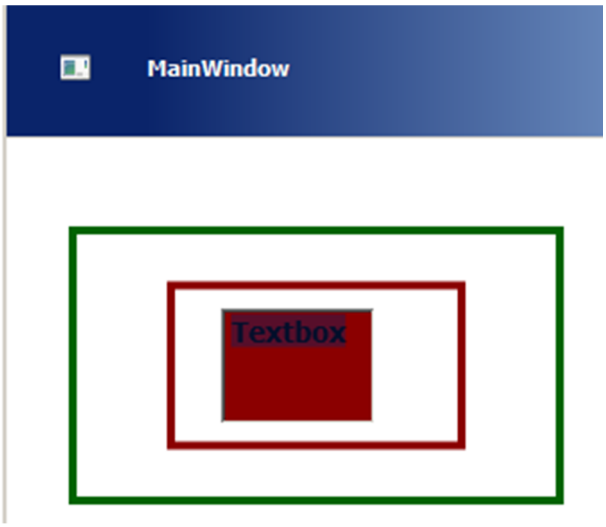
So the relative binding code with ancestor level 1 i.e. red looks as shown below. In case you are confused with any of the properties please refer to the previous table for more information.

```
<TextBox  Background="{Binding BorderBrush, RelativeSource={RelativeSource
FindAncestor, AncestorLevel=1, AncestorType={x:Type Border}}}"/>
```

So now the complete XAML with parent border element looks as shown in the below code.

```
<Border BorderBrush="DarkGreen"> <!-- Level 2 ->
<Border BorderBrush="DarkRed”> <!-- Level 1 ->

<TextBox  Background="{Binding BorderBrush, RelativeSource={RelativeSource
FindAncestor, AncestorLevel=1, AncestorType={x:Type Border}}}"/>

</Border>
</Border>
```

Now if you run the above XAML code the textbox is binded with the back ground color of the first border. If you change the ancestor level to 2 textbox background color will change to green.



PreviousData

```
<ItemsControl ItemsSource="{Binding}" Margin="10">
         <ItemsControl.ItemTemplate>
             <DataTemplate>
                 <StackPanel>
                     <TextBlock FontSize="14" FontWeight="bold"
                        Margin="20"
                        Text="{Binding Value}" Background="Aqua">
                     </TextBlock>
                     <TextBlock FontSize="14" FontWeight="bold"
                        Margin="20"
                        Text="{Binding
                        RelativeSource={RelativeSource PreviousData},
                                        Path=Value}" Background="Blue">

                     </TextBlock>

                 </StackPanel>
             </DataTemplate>
         </ItemsControl.ItemTemplate>
     </ItemsControl>

public class Item : INotifyPropertyChanged
  {
      private double _value;
```

```
        public double Value
        {
            get { return _value; }
            set { _value = value; OnPropertyChanged("Value"); }
        }

        #region INotifyPropertyChanged Members

        public event PropertyChangedEventHandler PropertyChanged;

        #endregion

        protected void OnPropertyChanged(string PropertyName)
        {
            if (null != PropertyChanged)
            {
                PropertyChanged(this,
                    new PropertyChangedEventArgs(PropertyName));
            }
        }
    }
    public class Items : ObservableCollection<Item>
    {
        public Items()
        {
            Add(new Item { Value = 80.23 });
            Add(new Item { Value = 126.17 });
            Add(new Item { Value = 130.21 });
            Add(new Item { Value = 115.28 });
            Add(new Item { Value = 131.21 });
            Add(new Item { Value = 135.22 });
            Add(new Item { Value = 120.27 });
            Add(new Item { Value = 110.25 });
            Add(new Item { Value = 90.20 });
        }
    }
```

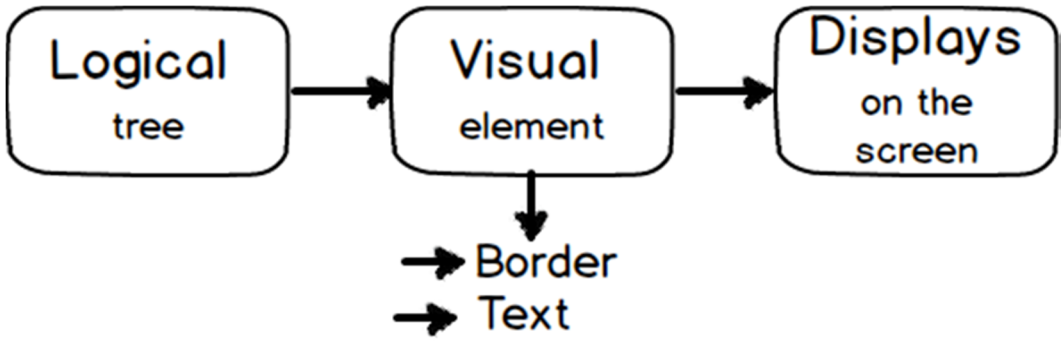
Explain the difference between visual and logical tree in WPF ?

WPF UI is represented in XAML which is a XML format. In XML elements are arranged in a hierarchal fashion. For example if you look at the below XAML (it has been downsized for simplicity) we have a Window element, the window element has a Grid control and the grid control has a button inside it.

```
<Window>
      <Grid>
            <Button..>
      </Grid>
```

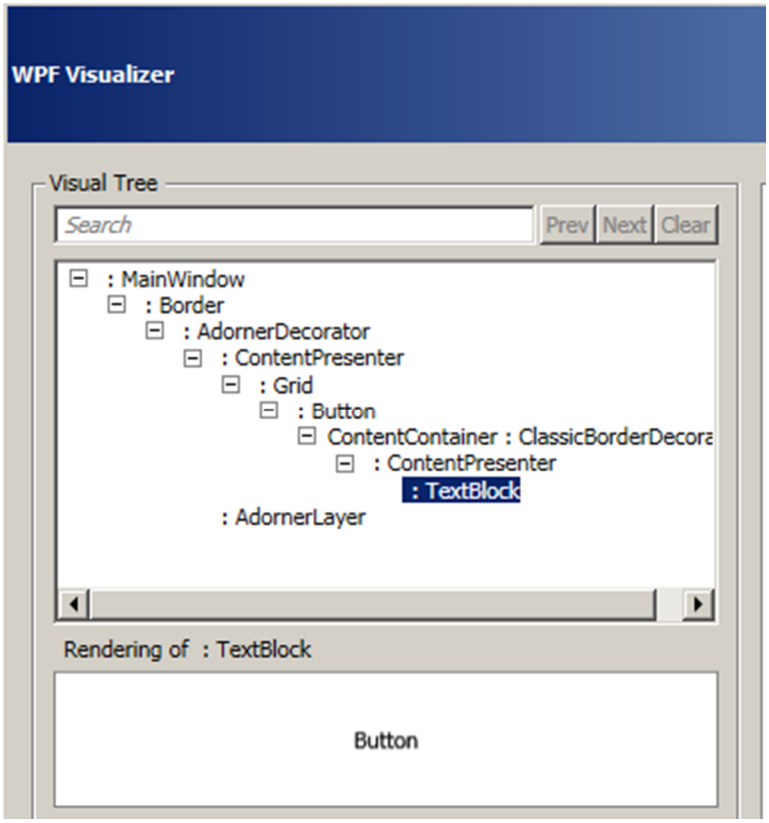
```
</Window>
```

So if you visualize the above XAML **logically** you can think that the button control is a child element of the Grid and the grid is the child element of the Window. This relationship between the elements which looks logical looking at the XAML is termed as "**Logical tree"**.

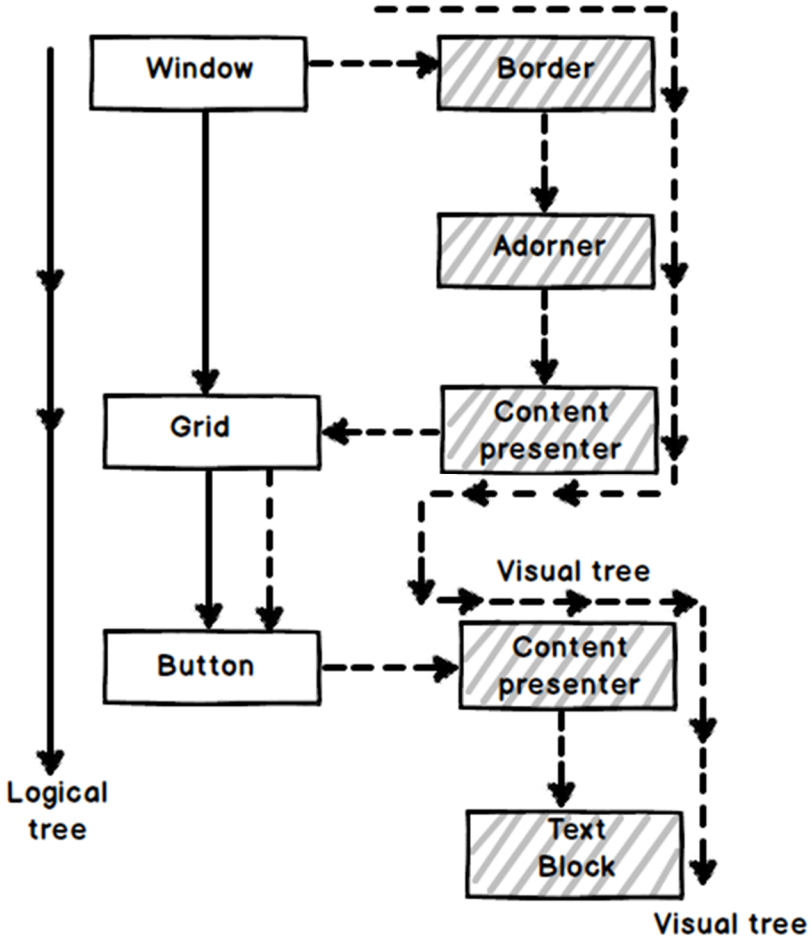


But now to display this Logical tree on to your screen you need lot of visual elements. like border, text etc. So when you add these visual elements to the logical tree that complete structure is termed as "**Visual Tree**".

Putting in simple words there is only tree in WPF but depending on how you view it these two trees are the outcome. If you use the WPF visualizer the above XAML tree looks something as shown in the below figure which is actually a complete visual tree.



In simple words whatever you see in your XAML is a logical tree and to make it display it uses the visual tree. Below is an in detail figure which shows the logical and visual tree of the above XAML. Logical tree is without the shades and with shades is the visual tree.



Why do we need to have these perspective of visual and logical tree in WPF ?

Visual tree and Logical tree are important when you work with WPF routed events.

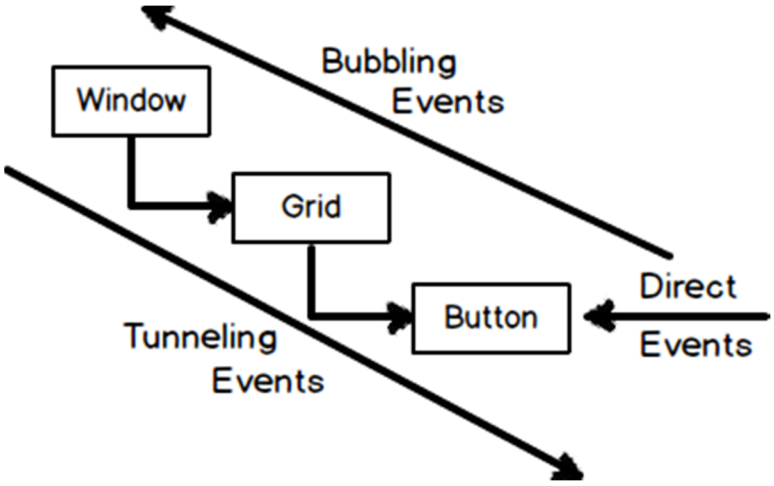
Explain routed events in WPF?

Routed events are those events which travel up or down the visual tree hierarchy. WPF events can be classified in to 3 types:-

Direct events: - In this case event is raised at the source and handled at the source itself like “MouseEnter” events.

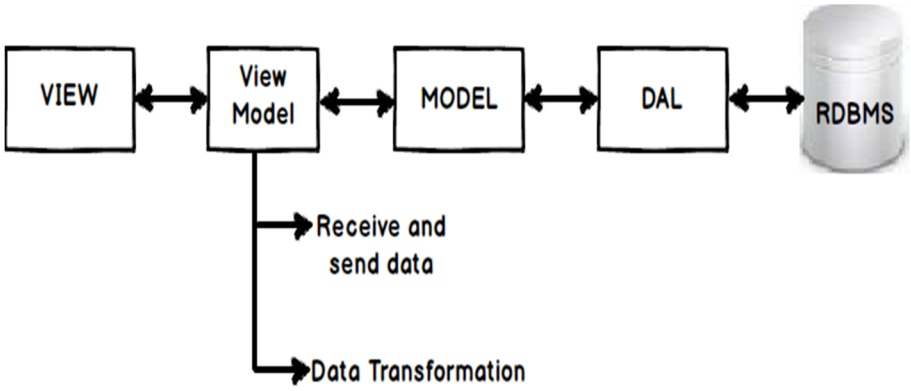
Bubbling events: - They travel up the visual tree hierarchy. For example “MouseDown” is a bubbling event.

Tunneling events: - These events travel down the visual tree hierarchy. “PreviewKeyDown” is a tunneling event.



What is MVVM?

MVVM is an architecture pattern where we divide the project in to three logical layers and every layer has its own responsibility.



Below are the three logical layers with explanation what they do:-

- View: - This layer handles responsibility of taking inputs from end user, positioning of controls, look and feel, design, visuals, colors etc.
- Model: - This layer represents your middle layer objects like customer, supplier etc. It handles business logic and interaction with data access layer.
- View Model: - View model as it says View plus model represents your user interface. You can also visualize this class as the behind code. This class is the bridge between model and view. It handles connection logic , data transformation logic and action mapping between model and view. For example it will have following types of logics as shown below :-

- Replicating and propagating data between views and models. When someone enters data in to UI or the model gets updated this layer will ensure that the propagation of data happens between these entities.
- Handling data transformation from view to model and vice versa. For example you have a model which has gender property with data as "M" for male and "F" for female. But on the View or UI you would like to display as a check box with true and false. This transformation logic is written in the view model class.
- View model also map UI actions to methods. For example you have "btn_Add" click event and when any one clicks on this button you would like to invoke "Customer.Add()" method from the customer class. This connection code is again a part if view model.

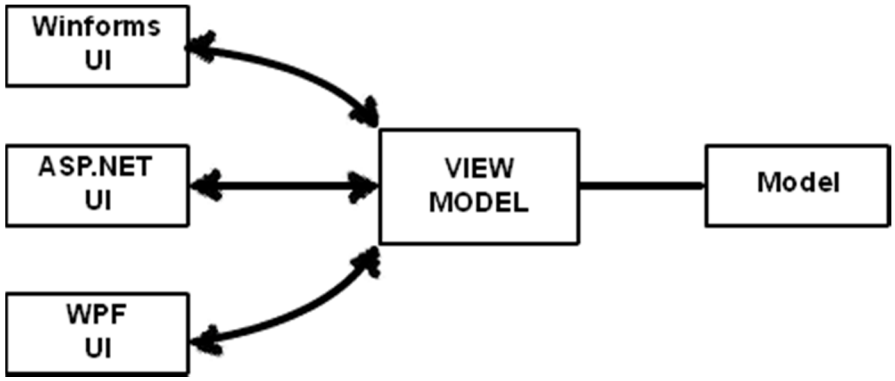
What are the benefits of MVVM?

Below are the benefits of MVVM pattern:-

Separation of concern: - As the project is divided in to layers, every layer handles its own responsibility. This leads to better maintenance because when we change one layer the other layer does not get affected.

Increased UI Reusability: - The whole point of MVVM is to remove behind code i.e. XAML.CS code. The problem with behind code is that it is tied up with a UI technology for example ASPX.CS code is tied up with the ASP page class, XAML.CS file is tied with WPF UI technology and so on. So we cannot use ASPX.CS behind code with WPF XAML UI.

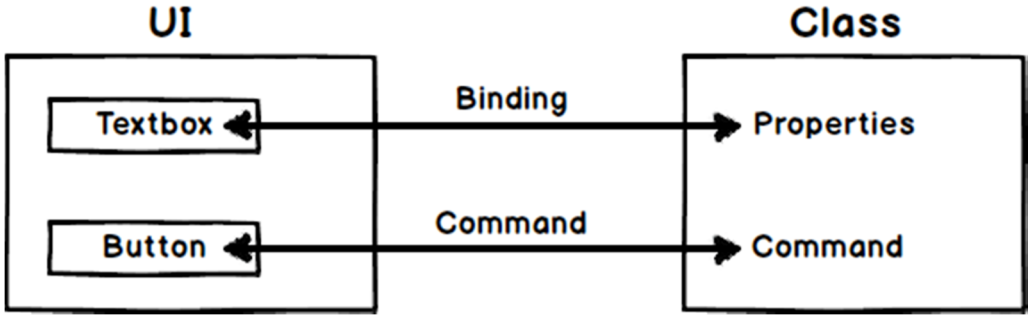
By moving the behind code to the view model class we can now use this with any UI technology.



Automated UI Unit testing: - View model class represents your UI. The properties of the class represent UI text boxes, combo boxes and the methods of the class represent action. Now as the UI is represented by the view model class, we can do automated UI testing using unit testing by creating the objects of view model class and doing the necessary asserts.

What is the importance of command and bindings in MVVM pattern?

MVVM in the most used architecture because of command and bindings facility provided by WPF. WPF MVVM is incomplete without command and bindings.



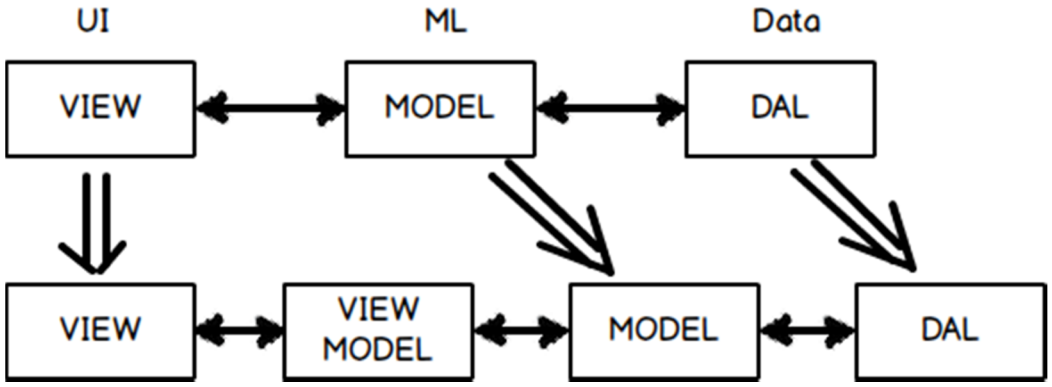
Command and bindings helps you to connect view (WPF UI) with view model class without writing lot of behind code. Binding connects the UI input elements (textbox, combo box etc.) with the view model class properties and the UI actions like button click, right click are connected to the methods of the class by commands.

```
Note :- Please refer previous questions to understand command and bindings.
```

What is the difference between MVVM and 3 layer architecture?

MVVM has an extra layer as compared to 3 layer architecture. In 3 layer architecture we have UI (view) , business logic (model) and Data access layer (DAL). In MVVM we have an extra layer in between view and model i.e. the view model class.

3 layer architecture complements MVVM architecture.

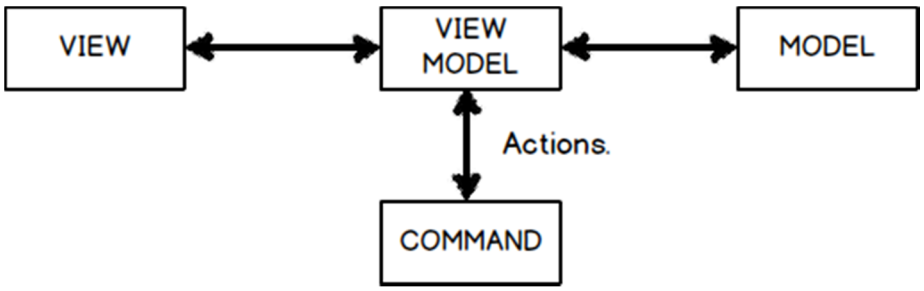


Explain delegate command?

First let us answer in short: - "*Delegate command makes a MVVM command class independent of the view model*". Now let's understand the long way.

In MVVM architecture view talks with the view model and view model talks with the model. When actions are sent from the view they are sent to WPF commands for handling the events. WPF commands invoked methods of view model internally.

In other words command needs reference of view model class.



If you see a typical WPF MVVM command it looks as shown below. You can see the "CustomerViewModel" class referenced inside the 'btnCommand" class. If you think with your eyes closed this reference of "CustomerViewModel" class inside the command is a problem. This will lead to tight coupling between command classes and view model.

If you visualize command it is nothing but click , double click , left mouse click , drag and drop etc. It's an ACTION created by the user. Now wouldn't be great if we can just attach this command with any view model. So like click event gets connected with "CustomerViewModel" or "SupplierViewModel".

This is achieved by using delegate command.

```
public class btnCommand : ICommand
{
        Private CustomerViewModel Viewobj = new CustomerViewModel();


        public btnCommand(CustomerViewModel obj)
        {
            Viewobj = obj;
        }
        public bool CanExecute(object parameter) // When he should execute
        {
            return Viewobj.IsValid();
        }
```

```
        public void Execute(object parameter) // What to execute
        {
            ViewObj.Add();
        }
    }
```

To decouple the view model class from command we can use delegates i.e. “Action” and “Func”. If you see the command class we need only two things “WhattoExecute” and “WhentoExecute”. So how about passing these methods as generic delegates. You can see the constructor of “btnCommand” takes two delegates one what to execute and when to execute.

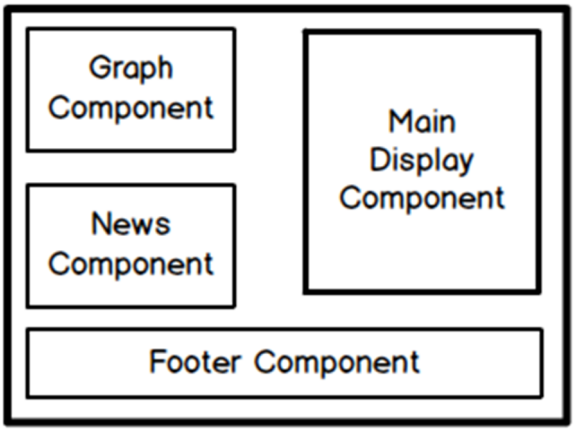
You can see in the below code the “btnCommand” class has no reference of the view model class but has references to delegates which are just abstract pointer to functions / methods. So this command class can now be attached with any view model class. This approach is termed as “Delegate command”.

```
public class btnCommand : ICommand // Step 1 :- Create command
    {
        private Action WhattoExecute;
        private Func<bool> WhentoExecute;
        public btnCommand(Action What , Func<bool> When)
        {
            WhattoExecute = What;
            WhentoExecute = When;
        }
        public bool CanExecute(object parameter) // When he should execute
        {
            return WhentoExecute();
        }



        public void Execute(object parameter) // What to execute
        {

            WhattoExecute();
        }
    }
```

What is PRISM?

PRISM is a framework to develop composite application in WPF and Silverlight. Composite applications are built using composition. In other words rather than building application from scratch we take prebuilt components, assemble them together and create the application.

Take the below example of simple WPF UI. You can see it has lots of sections. Now rather than building the whole UI as one big unit, we can develop all these section as independent unit. Later by using PRISM we can compose WPF UI by taking all these independent units.



What are benefits of PRISM?

Modular development:- As we are developing components as independent units we can assign these units to different developers and do modular parallel development. With parallel development project will be delivered faster.

High reusability:- As the components are developed in individual units we can plug them using PRISM and create composed UI in an easy way.

How are individual units combined in to a single unit ?

PRISM uses dependency injection for the same. Please see design pattern section to read about DI (dependency injection). We can do DI by two way's by using unity application block or MEF (managed extensibility framework).

Does PRISM do MVVM?

The prime focus of PRISM is modular development and not MVVM. But it does have readymade classes like delegate command which can help us reduce MVVM code. But please note the main goal of PRISM was not MVVM.

Is PRISM a part of WPF?

No, PRISM is a separate installation.

What is expression blend?

Expression blend is a designing tool for WPF and Silverlight application.