Introduction

In this section we will run through important WPF interview questions asked during .NET interviews. I know the list is not complete but wanted to at least start with something and then keep adding as time goes.

If you feel I should be covering some more question please enter the question in the comments below.

Happy job hunting  .

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?

Hide   Copy Code

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)

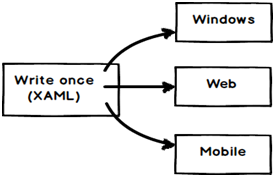
F - Fast execution ( Hardware acceleration)

G - Graphic hardware independent ( resolution independent)

I would strongly suggest you to see the below videowhich explains all the above A to G points practically.

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.

Hide   Copy Code

<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 “” and the restaurant table element qualified with “”.

Hide   Copy Code

<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.

Hide   Copy Code

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

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

Hide   Copy Code

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.

Hide   Copy Code

<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.

Hide   Copy Code

<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.

Hide   Copy Code

<StackPanel x:Name="myStack" />

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

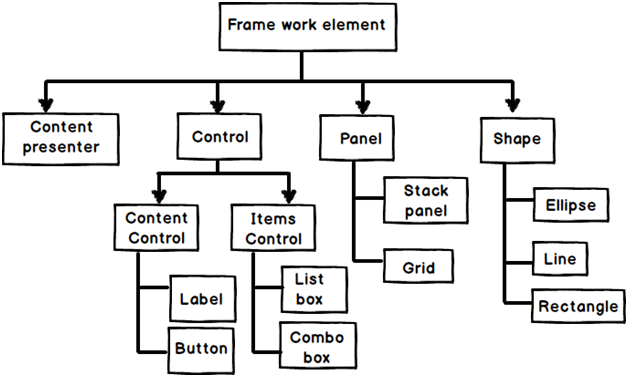
There is no difference between “x:name” and “name” , “name” is short hand of “x:name”. But in classes where you do not find “name” property ( this is a rare situations) we need to use “x:name” property.

What are the different kinds of controls in WPF?

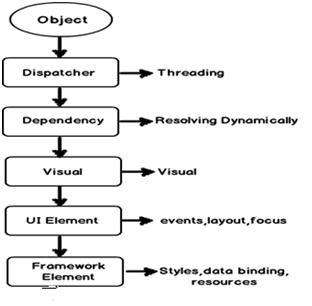
WPF controls can be categorized in to four categories:-

* **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 ?

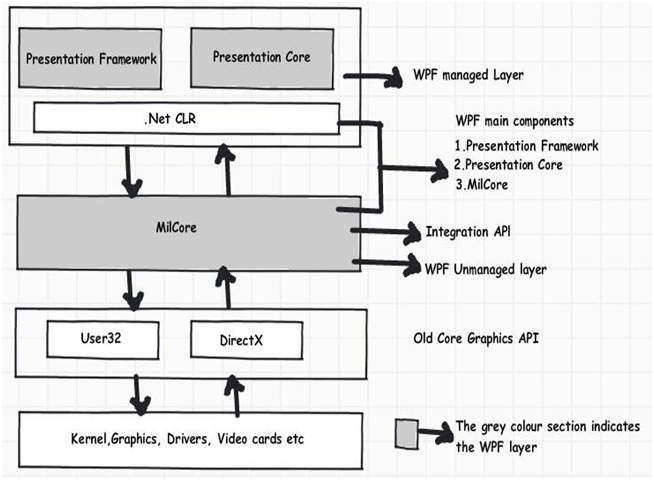
No,XAML is not meant only for WPF.XAML is a XML-based language and it had various variants.

**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?



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 various ways of doing alignment in WPF?

There are five ways of doing alignment in WPF:-

**Grid:-** In Grid alignment we divide the screen in to static rows and columns like HTML tables and position elements in those rows and column area.

Hide   Copy Code

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

</Grid.RowDefinitions>

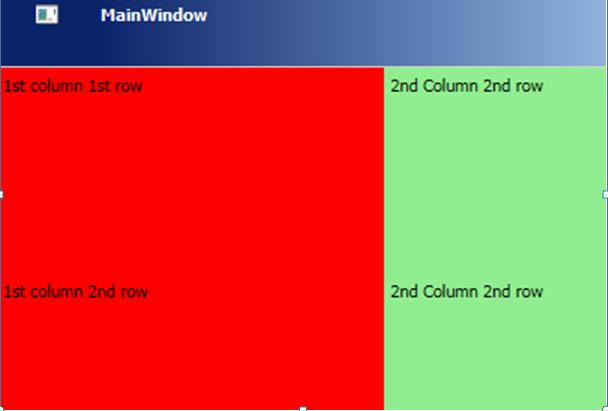
<Label Grid.Column="0" Grid.Row="0" Background="Red">1st column 1st row </Label>

<Label Grid.Column="1" Grid.Row="0" Background="LightGreen">2nd Column 2nd row</Label>

<Label Grid.Column="0" Grid.Row="1" Background="Red">1st column 2nd row</Label>

<Label Grid.Column="1" Grid.Row="1" Background="LightGreen">2nd Column 2nd row</Label>

</Grid>



**Stack panel :-** Arranges control in vertical or horizontal format.

Hide   Copy Code

<StackPanel Orientation="Vertical">

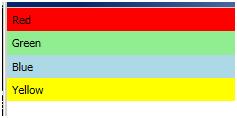
<Label Background="Red">Red </Label>

<Label Background="LightGreen">Green </Label>

<Label Background="LightBlue">Blue </Label>

<Label Background="Yellow">Yellow </Label>

</StackPanel>



**WrapPanel :-**Aligns elements in a line until the border is hit , then wraps in to the next line.

Hide   Copy Code

<WrapPanel Orientation="Horizontal">

<Label Width="125" Background="Red">Red 1</Label>

<Label Width="100" Background="LightGreen">Green 1</Label>

<Label Width="125" Background="LightBlue">Blue 1</Label>

<Label Width="50" Background="Yellow">Yellow 1</Label>

<Label Width="150" Background="Orange">Orange 1</Label>

<Label Width="100" Background="Red">Red 2</Label>

<Label Width="150" Background="LightGreen">Green 2</Label>

<Label Width="75" Background="LightBlue">Blue 2</Label></WrapPanel>



**Dock Panel: -**Aligns controls in five different regions: top, bottom, left, right and center.

Hide   Copy Code

<DockPanel>

<Label DockPanel.Dock="Top" Height="100" Background="Red">Top 1</Label>

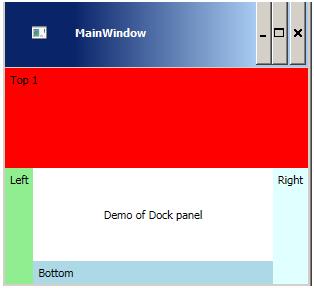
<Label DockPanel.Dock="Left" Background="LightGreen">Left</Label>

<Label DockPanel.Dock="Right" Background="LightCyan">Right</Label>

<Label DockPanel.Dock="Bottom" Background="LightBlue">Bottom</Label>

<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"> Demo of Dock panel</TextBlock>

</DockPanel>



**Canvas :-**Positions elements absolutely using co-ordinates.

Hide   Copy Code

<Canvas Margin="273,130,144,99">

<TextBlock> Canvas position </TextBlock>

</Canvas>



What are resources in WPF?

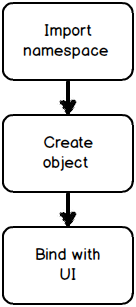
Resources are objects referred in WPF XAML. In C# code when we create an object we do the following three steps :-

Hide   Copy Code

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.

Hide   Copy 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”.

Hide   Copy Code

<Window.Resources>

<custns:Customer x:Key="custobj"/>

</Window.Resources>

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

Hide   Copy Code

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.

Hide   Copy Code

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

Explain the difference between static and dynamicresource?

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.

Hide   Copy Code

<Window.Resources>

<SolidColorBrush Color="LightBlue" x:Key="buttonBackground" />

</Window.Resources>

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

Hide   Copy Code

<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.

Hide   Copy Code

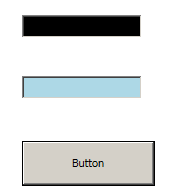
private void Button\_Click(object sender, RoutedEventArgs e)

{

Resources["buttonBackground"] = Brushes.Black;

}

Below is the output of the same.

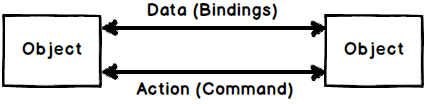


When should we use static resource over dynamic resource ?

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?

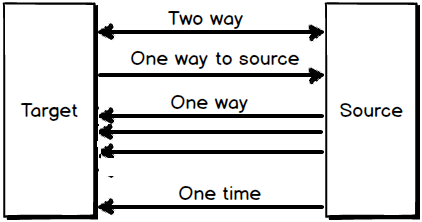
All the above 4 things define how data will flow between target and source objects when WPF binding is applied.

**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.

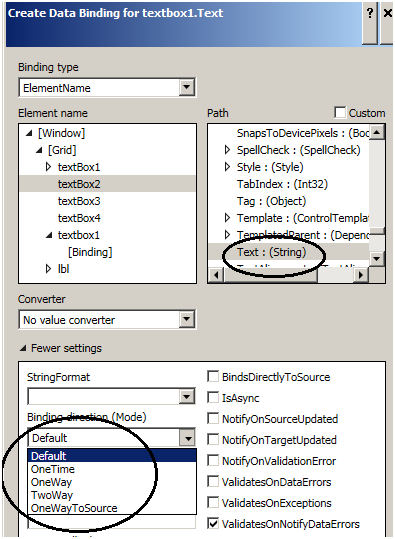
**One time: -** Data flows only for the first time from source to target and after that no communication happens.



Below is an easy tabular representation to memorize the same.

One way to source

|  |  |
| --- | --- |
|  | **Data flow** |
| **Bindings** | **Target to Source** | **Source 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.

Hide   Copy Code

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.

Hide   Copy Code

<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:-

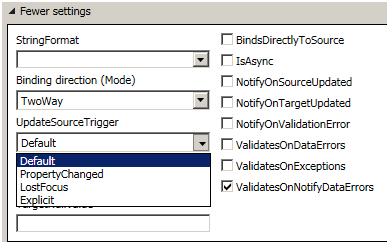
* Default: - If it’s a text property then data is updated during lost focus and for normal properties data updates in property change event.
* PropertyChanged: - In this setting data is updated as soon as the value is changed.
* LostFocus: - In this setting data is updated as soon as lost focus event 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.

Hide   Copy Code

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

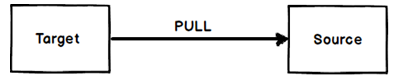
binding.UpdateSource();

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.

Hide   Copy Code

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

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

Hide   Copy Code

public class clsCounter : INotifyPropertyChanged

{

private int \_Counter=0;

public int Counter

{

get { return \_Counter; }

}

public void Increment()

{

\_Counter++;

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

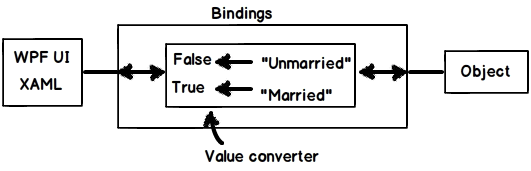
}

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”.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

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";

}

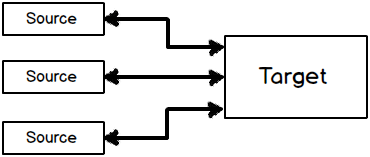
}

}

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.

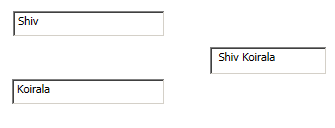
Explain multi binding and multivalue converters in WPF?

“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 “FirstNameLastName” ( As shown in the below figure).

Also vice versa if we type in the third text box “FirstNameLastName” 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 “FirstNameLastName” 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”.

Hide   Copy Code

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.

Hide   Copy Code

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.

Hide   Copy Code

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.

Hide   Copy Code

<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.

Hide   Copy Code

<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.

Hide   Copy Code

<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.

Hide   Copy Code

<Border BorderBrush="DarkGreen">

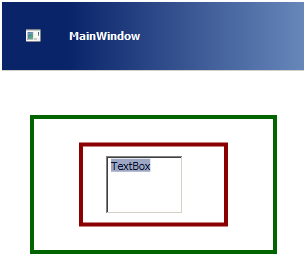
<Border BorderBrush="DarkRed”>

<TextBox />

</Border>

</Border>

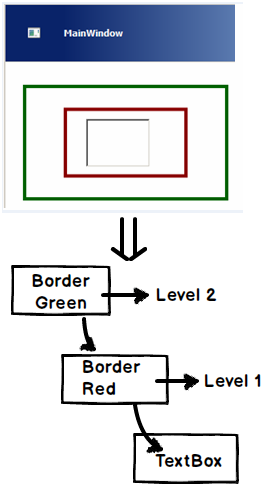
Below is how the WPF application looks like when it runs.



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.

Hide   Copy Code

<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.

Hide   Copy Code

<Border BorderBrush="DarkGreen"><!-- *Level 2 ->*

*<Border BorderBrush="DarkRed”><!-- Level 1 ->*

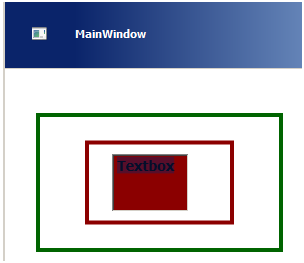
*<TextBox removed="{Binding BorderBrush, RelativeSource={RelativeSource FindAncestor, AncestorLevel=1, AncestorType=*

*{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

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

<ItemsControlItemsSource="{Binding}" Margin="10">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel>

<TextBlockFontSize="14"FontWeight="bold"

Margin="20"

Text="{Binding Value}" removed="Aqua">

</TextBlock>

<TextBlockFontSize="14"FontWeight="bold"

Margin="20"

Text="{Binding

RelativeSource={RelativeSourcePreviousData},

Path=Value}" removed="Blue">

</TextBlock>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

publicclassItem : INotifyPropertyChanged

{

privatedouble \_value;

publicdouble Value

{

get { return \_value; }

set { \_value = value; OnPropertyChanged("Value"); }

}

#regionINotifyPropertyChanged Members

publiceventPropertyChangedEventHandlerPropertyChanged;

#endregion

protectedvoidOnPropertyChanged(stringPropertyName)

{

if (null != PropertyChanged)

{

PropertyChanged(this,

newPropertyChangedEventArgs(PropertyName));

}

}

}

publicclassItems : ObservableCollection<Item>

{

public Items()

{

Add(newItem { Value = 80.23 });

Add(newItem { Value = 126.17 });

Add(newItem { Value = 130.21 });

Add(newItem { Value = 115.28 });

Add(newItem { Value = 131.21 });

Add(newItem { Value = 135.22 });

Add(newItem { Value = 120.27 });

Add(newItem { Value = 110.25 });

Add(newItem { 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.

Hide   Copy Code

<Window>

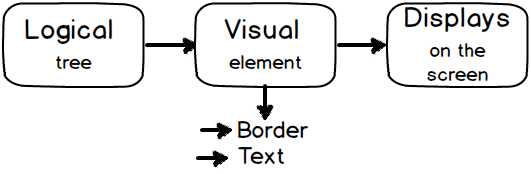
<Grid>

<Button..>

</Grid>

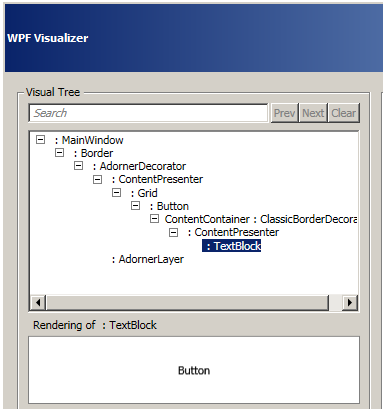
</Window>

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**”.

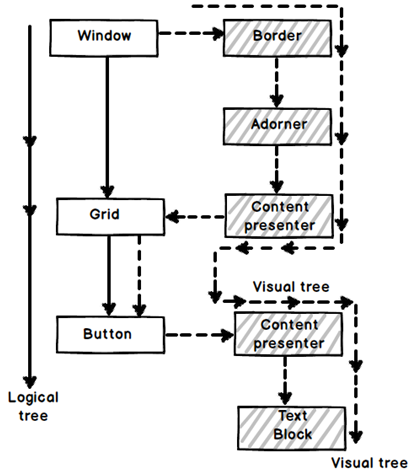


But now to display this Logical tree on to your screen you need lot of visual elements. like border, text etc. So when you add these visual elements to the logical tree that complete structure is termed as “**Visual Tree**”.

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.

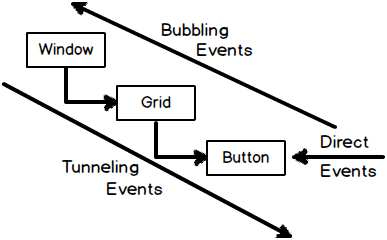
Explain routed events in WPF?

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 are styles in WPF?

WPF style helps to define look and feel ( color , fonts, alignments etc) from a central location so that we can have ease of maintenance.For more details on WPF styles please check this link out which also has [simple sample WPF examples](http://dotnetinter.livejournal.com/89700.html)

What are style triggers?

Many times we would want a style property to execute under certain conditions. Trigger’s help you to define condition under which the style values will be set.

For instance in the below XAML code we have style trigger defined for button which set’s background color to “Red” if there is focus on the button or else the back ground color is set to “Aqua”.

Hide   Copy Code

<Style x:Key="myStyle" TargetType="Button">

<Style.Triggers>

<Trigger Property="IsFocused" Value="True">

<Setter Property="Background" Value="Red"/>

</Trigger>

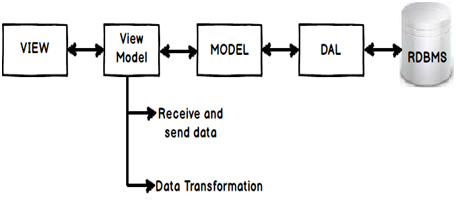
</Style.Triggers>

<Setter Property="Background" Value="Aqua" />

</Style>

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.

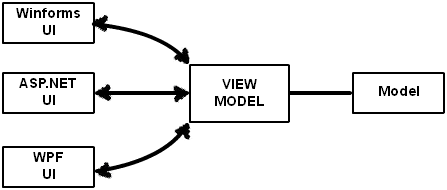
What are the benefits of MVVM?

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.

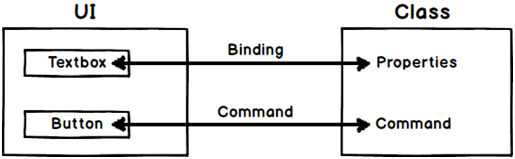
By moving the behind code to the view model class we can now use this with any UI technology.



**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.

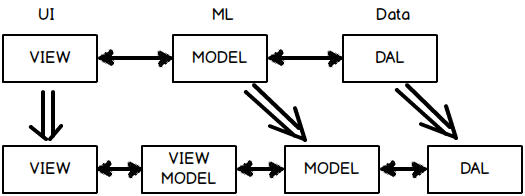
Hide   Copy Code

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.

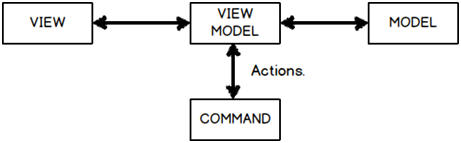


Explain delegate command?

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.

Hide   Copy Code

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”.

Hide   Copy Code

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.

<http://www.codeproject.com/Articles/744082/WPF-Interview-questions-with-answers>

<https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements>

# WPF: Versions, History and Major

# Recently I was looking for WPF major versions and its progression so far. To my surprise there was no resource providing a compact look on WPF versions on internet. Then I put some efforts in this regard and came up with this article.

[**Quick Look on WPF Versions (pdf)**](https://www.codeproject.com/KB/WPF/1035800/Quick_Look_at_WPF_Versions_and_Major_Enhancements.zip)

## Introduction

Windows Presentation Foundation (WPF) is a framework to create stunning GUI applications. It is so popular in .NET Community that nearly 10% of all newly created projects are WPF as described [here on MSDN](http://blogs.msdn.com/b/dotnet/archive/2014/11/12/the-roadmap-for-wpf.aspx).  
  
But when I searched for its major versions and enhancements respective version brought, I could not find concise resource. Unfortunately its [Wikipedia page](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation) looks like not up to date. Even on MSDN there is no webpage providing such summary. So I collected information from various resources and presented in this article. I have provided links to those resources wherever needed and more in [*References*](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#References) section of this article as those would be pointers for further study. My humble thanks to authors of those great blogs / articles.  
  
***Note:***This article for intermediate/advance level who are well versed with WPF. For introduction to WPF, please visit to [MSDN Site](https://msdn.microsoft.com/en-us/library/ms754130(v=vs.100).aspx) and [Wikipedia](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation).

## Contents

This article is having following sections:

* [WPF Versions and Major Enhancements](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-Versions-And-Major-Enhancements)
* [Path for WPF Version Assemblies](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#Path-For-WPF-Version-Assemblies)
* [WPF 3 - Initial Release](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-3-Initial-Release)
* [WPF 3.5 Enhancements](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-3-5-Enhancements)
* [WPF 3.5 SP1 Enhancements](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-3-5-SP1-Enhancements)
* [WPF 4 Enhancements](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-4-Enhancements)
* [WPF 4.5 Enhancements](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-4-5-Enhancements)
* [WPF 4.6 Enhancements](https://www.codeproject.com/Articles/1035800/WPF-Versions-History-and-Major-Enhancements#WPF-4-6-Enhancements)

## WPF Versions and Major Enhancements

Following table gives a concise view of WPF Versions, timeline and major enhancements:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **WPF Version** | **Release (YYYY-MM)** | **.NET Version** | **Visual Studio Version** | **Major Features** |
| 3.0 | 2006-11 | 3.0 | N/A | Initial Release. WPF development can be done with VS 2005 (released in Nov 2005) too with few additions as described [here](http://blogs.msdn.com/b/tims/archive/2006/12/20/building-a-perfect-wpf-developer-workstation.aspx). |
| 3.5 | 2007-11 | 3.5 | VS 2008 | Changes and improvements in: Application model, data binding, controls, documents, annotations, and 3-D UI elements. |
| 3.5 SP1 | 2008-08 | 3.5 SP1 | N/A | Native splash screen support, New WebBrowser control, DirectX pixel shader support. Faster startup time and improved performance for Bitmap effects. |
| 4.0 | 2010-04 | 4.0 | VS 2010 | New controls: Calendar, DataGrid, and DatePicker. Multi-Touch and Manipulation |
| 4.5 | 2012-08 | 4.5 | VS 2012 | New Ribbon control New INotifyDataErrorInfo interface |
| 4.5.1 | 2013-10 | 4.5.1 | VS 2013 | No Major Change |
| 4.5.2 | 2014-05 | 4.5.2 | N/A | No Major Change |
| 4.6 | 2015-07 | 4.6 | VS 2015 | Transparent child window support HDPI and Touch improvements |

## Path for WPF Version Assemblies

The reference assemblies can be found at folder path:  
“C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework”.  
There you will get folders for each .NET framework version installed from .NET 3.5 onwards. In each folder, you will get ***PresentationCore.dll*** and ***PresentationFramework.dll*** which comes with respective .NET Version. You can check the difference in their build versions by looking into “Details” tab of properties windows for any DLL. For 3.0 and Silverlight, refer to path:  
"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework".  
For more information, please have a look at [this link](http://stackoverflow.com/questions/12070518/where-is-the-net-framework-4-5-directory).

## WPF 3 - Initial Release

The journey of WPF formally started with its first released in 2006. As WPF was released as a part of .NET Framework 3.0 so first version of WPF is also called WPF 3.0. It was having code-name as "Avalon" while its development. Differentiating basic features of WPF are:

1. **Employing DirectX:** In order to obtain GPU-accelerated rendering, WPF renders and presents graphics through the DirectX pipeline (rather than relying on the older GDI subsystem). DirectX allows to offload graphics tasks to the GPU. This frees up the CPU to do other work and thus increases app performance.
2. **Declarative UI through XAML:** Extensible Application Markup Language (XAML) is an XML-based markup language. XAML is the language behind the visual presentation of WPF applications.
3. **Dependency Properties:** WPF introduces a new type of property called a dependency property. A dependency property depends on multiple providers for determining its value at any point in time. The value of a DependencyProperty is resolved dynamically. Major advantages of dependency properties are reduced memory footprint, value inheritance and change notification.
4. **Data binding:** Data binding is the process that establishes a connection between the application UI and the business logic. Using data binding we can take data from almost any property of any object and bind it to almost any other dependency property of another object.
5. **Templates and Styles:** WPF gives us an extensibility model based on templates and styles, allowing us to perform complex changes and redefinitions of the visual appearance of a control, without having to create a new one. You can change the structure and appearance of a control by modifying the ***ControlTemplate*** of that control. A Style object is a collection of values representing properties for the specified control.

## WPF 3.5 Enhancements

1. **Improvements to the application model:**
   * XBAPs can now run in Firefox.
   * Cookies can be shared between XBAPs and Web applications from the same site of origin.
   * Improved XAML IntelliSense experience for higher productivity and Expanded localization support.
   * Comprehensive add-in support for supporting nonvisual and visual add-ins from standalone applications and XAML browser applications (XBAPs).
2. **Ability to cache images:**You now have the ability to cache images that are downloaded over HTTP to the local Microsoft Internet Explorer temporary file cache, so that subsequent requests for the image come from local disk, rather than the Internet. Depending on the sizes of your images, this can be a significant network performance improvement.
3. **Interactive 3-D Graphics:**new features have been added to the 3-D object model:
   * Input, Focus, and Eventing Support in 3-D.
   * Interactive 2-D Content on 3-D.
   * New Transformation Services.
4. **Enhanced Data Binding:**
   * A new debugging mechanism makes it easier to debug data bindings.
   * The data model enables validation on the business layer by providing support for the IDataErrorInfo interface. In addition, the validation model now supports using property syntax to set validation rules.
   * The data binding model now supports LINQ and XLINQ.
5. **Text Improvements** in Controls: Enhancements in RichTextBox and TextBoxBase.
6. **Documents Improvement:** FlowDocumentPageViewer, FlowDocumentScrollViewer, and FlowDocumentReader each have a new public property called Selection. The property gets the TextSelection that represents the selected content in the document.
7. **Annotations Improvement:** The annotations framework now exposes the capabilities for matching annotations with the corresponding annotated objects. A new interface, IAnchorInfo, has been added. In addition, a new method, GetAnchorInfo, which returns an IAnchorInfo object, has been added to the AnnotationHelper class.

For more information, please have a look at [What's New in WPF Version 3.5](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.90).aspx).

## WPF 3.5 SP1 Enhancements

1. **Improved performance** for Bitmap effects and faster startup.
2. New **WebBrowser control.**
3. Additional functionality includes better support for line of business applications,**native splash screen support, DirectX pixel shader support.**

For more information, please have a look at [What's New in the .NET Framework Version 3.5 SP1](https://msdn.microsoft.com/en-us/library/cc713697(v=vs.90).aspx).

## WPF 4 Enhancements

1. **New controls**, including Calendar, DataGrid, and DatePicker.
2. **Multi Touch and Manipulation** enables you to create applications that receive input from multiple touches simultaneously on Windows 7.
3. Graphics and animation supports layout rounding, Pixel Shader version 3.0, cached composition, and easing functions.
4. New types in the **System.Windows.Shell** namespace enable you to communicate with the Windows 7 taskbar and pass data to the Windows shell.
5. VisualStateManager supports changing states of controls.
6. Text has improved text rendering and supports customizing the caret color and selection color in text boxes.
7. **Binding** is supported on the Command property of an InputBinding, dynamic objects, and the Text property.
8. XAML browser applications (XBAPs) support communication with the Web page and support full-trust deployment.

For more information, please have a look at [What's New in WPF Version 4](https://msdn.microsoft.com/en-us/library/vstudio/bb613588(v=vs.100).aspx).

## WPF 4.5 Enhancements

1. New **Ribbon control**, which enables you to implement a ribbon user interface that hosts a Quick Access Toolbar, Application Menu, and tabs.
2. The new **INotifyDataErrorInfo** interface, which supports synchronous and asynchronous data validation.
3. Ability to set the amount of time that should **elapse between property changes and data source updates**.
4. Improved support for implementing **weak event patterns**. Also, events can now accept markup extensions.
5. New features for the VirtualizingPanel and Dispatcher classes.
6. Data binding to static properties, data binding to custom types that implement the ICustomTypeProvider interface, and retrieval of data binding information from a binding expression.
7. **Ability to check** whether the data context for an item container is disconnected.
8. **Improved performance**when displaying large sets of grouped data, and by accessing collections on non-UI threads.

For more information, please have a look at [What's New in .NET 4.5](https://msdn.microsoft.com/en-us/library/ms171868(v=vs.110).aspx#core).

## WPF 4.6 Enhancements

1. **HDPI improvements:** HDPI support in WPF is now better in the .NET Framework 4.6. Changes have been made to layout rounding to reduce instances of clipping in controls with borders.
2. **Transparent child window support:** WPF in the .NET Framework 4.6 supports transparent child windows in Windows 8.1 and above. This allows you to create non-rectangular and transparent child windows in your top-level windows. You can enable this feature by setting the HwndSourceParameters.UsesPerPixelTransparency property to true.
3. **Touch Improvements:** In older version, touch produces unpredictable behavior have been addressed in the .NET Framework 4.6. The double tap threshold for Windows Store applications and WPF applications is now the same in Windows 8.1 and above.

For more information, please have a look at [What's New in .NET 4.6](https://msdn.microsoft.com/en-us/library/ms171868(v=vs.110).aspx#v46).

## Conclusion

I hope this article will help you to have a quick look and understand the differences between various versions of WPF. Your queries and comments are most welcome for improvements. Thanks.

**What is dispatcher in WPF**

This class provides a property named **Dispatcher** that returns the **Dispatcher** object associated with the **WPF** element. The **Dispatcher** class is used to perform work on his attached thread. It has a queue of work items and it is in charge of executing the work items on the **dispatcher** thread

# What's New in WPF Version 4.5

<https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx>

Thistopic contains information about new and enhanced features in Windows Presentation Foundation (WPF) version 4.5.

This topic contains the following sections:

* [Ribbon control](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#ribbon_control)
* [Improved performance when displaying large sets of grouped data](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#grouped_virtualization)
* [New features for the VirtualizingPanel](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#VirtualizingPanel)
* [Binding to static properties](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#static_properties)
* [Accessing collections on non-UI Threads](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#xthread_access)
* [Synchronously and Asynchronously validating data](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#INotifyDataErrorInfo)
* [Automatically updating the source of a data binding](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#delay)
* [Binding to types that Implement ICustomTypeProvider](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#ICustomTypeProvider)
* [Retrieving data binding information from a binding expression](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#binding_state)
* [Checking for a valid DataContext object](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#DisconnectedSource)
* [Repositioning data as the data's values change (Live shaping)](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#live_shaping)
* [Improved Support for Establishing a Weak Reference to an Event](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#weak_event_pattern)
* [New methods for the Dispatcher class](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#async)
* [Markup Extensions for Events](https://msdn.microsoft.com/en-us/library/bb613588(v=vs.110).aspx#events_markup_extenions)

## Ribbon control

WPF 4.5 ships with a [Ribbon](https://msdn.microsoft.com/en-us/library/system.windows.controls.ribbon.ribbon(v=vs.110).aspx) control that hosts a Quick Access Toolbar, Application Menu, and tabs. For more information, see the [Ribbon Overview](https://msdn.microsoft.com/en-us/library/bb386097(v=vs.110).aspx).

## Improved performance when displaying large sets of grouped data

UI virtualization occurs when a subset of user interface (UI) elements are generated from a larger number of data items based on which items are visible on the screen. The [VirtualizingPanel](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel(v=vs.110).aspx) defines the [IsVirtualizingWhenGrouping](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.isvirtualizingwhengrouping(v=vs.110).aspx) attached property that enables UI Virtualization for grouped data. For more information about grouping data, see How to: Sort and Group Data Using a View in XAML. For more information about virtualizing grouped data, see the [IsVirtualizingWhenGrouping](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.isvirtualizingwhengrouping(v=vs.110).aspx) attached property.

## New features for the VirtualizingPanel

1. You can specify whether a [VirtualizingPanel](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel(v=vs.110).aspx), such as the [VirtualizingStackPanel](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingstackpanel(v=vs.110).aspx), displays partial items by using the [ScrollUnit](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.scrollunit(v=vs.110).aspx) attached property. If [ScrollUnit](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.scrollunit(v=vs.110).aspx) is set to [Item](https://msdn.microsoft.com/en-us/library/system.windows.controls.scrollunit(v=vs.110).aspx), the [VirtualizingPanel](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel(v=vs.110).aspx) will only display items that are completely visible. If [ScrollUnit](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.scrollunit(v=vs.110).aspx) is set to [Pixel](https://msdn.microsoft.com/en-us/library/system.windows.controls.scrollunit(v=vs.110).aspx), the [VirtualizingPanel](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel(v=vs.110).aspx) can display partially visible items.
2. You can specify the size of the cache before and after the viewport when the [VirtualizingPanel](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel(v=vs.110).aspx) is virtualizing by using the [CacheLength](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.cachelength(v=vs.110).aspx) attached property. The cache is the amount of space above or below the viewport in which items are not virtualized. Using a cache to avoid generating UI elements as they’re scrolled into view can improve performance. The cache is populated at a lower priority so that the application does not become unresponsive during the operation. The [VirtualizingPanel.CacheLengthUnit](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.cachelengthunit(v=vs.110).aspx) property determines the unit of measurement that is used by [VirtualizingPanel.CacheLength](https://msdn.microsoft.com/en-us/library/system.windows.controls.virtualizingpanel.cachelength(v=vs.110).aspx).

## Binding to static properties

You can use static properties as the source of a data binding. The data binding engine recognizes when the property's value changes if a static event is raised. For example, if the class SomeClass defines a static property called MyProperty, SomeClass can define a static event that is raised when the value of MyProperty changes. The static event can use either of the following signatures.

* public static event EventHandler MyPropertyChanged;
* public static event EventHandler<PropertyChangedEventArgs> StaticPropertyChanged;

Note that in the first case, the class exposes a static event named PropertyNameChanged that passes [EventArgs](https://msdn.microsoft.com/en-us/library/system.eventargs(v=vs.110).aspx) to the event handler. In the second case, the class exposes a static event named StaticPropertyChanged that passes [PropertyChangedEventArgs](https://msdn.microsoft.com/en-us/library/system.componentmodel.propertychangedeventargs(v=vs.110).aspx) to the event handler. A class that implements the static property can choose to raise property-change notifications using either method.

## Accessing collections on non-UI Threads

WPF enables you to access and modify data collections on threads other than the one that created the collection. This enables you to use a background thread to receive data from an external source, such as a database, and display the data on the UI thread. By using another thread to modify the collection, your user interface remains responsive to user interaction.

## Synchronously and Asynchronously validating data

The [INotifyDataErrorInfo](https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifydataerrorinfo(v=vs.110).aspx) interface enables data entity classes to implement custom validation rules and expose validation results asynchronously. This interface also supports custom error objects, multiple errors per property, cross-property errors, and entity-level errors. For more information, see [INotifyDataErrorInfo](https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifydataerrorinfo(v=vs.110).aspx).

## Automatically updating the source of a data binding

If you use a data binding to update a data source, you can use the [Delay](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingbase.delay(v=vs.110).aspx) property to specify an amount of time to pass after the property changes on the target before the source updates. For example, suppose that you have a [Slider](https://msdn.microsoft.com/en-us/library/system.windows.controls.slider(v=vs.110).aspx) that has its [Value](https://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.rangebase.value(v=vs.110).aspx) property data two-way bound to a property of a data object and the [UpdateSourceTrigger](https://msdn.microsoft.com/en-us/library/system.windows.data.binding.updatesourcetrigger(v=vs.110).aspx) property is set to [PropertyChanged](https://msdn.microsoft.com/en-us/library/system.windows.data.updatesourcetrigger(v=vs.110).aspx). In this example, when the user moves the [Slider](https://msdn.microsoft.com/en-us/library/system.windows.controls.slider(v=vs.110).aspx), the source updates for each pixel that the [Slider](https://msdn.microsoft.com/en-us/library/system.windows.controls.slider(v=vs.110).aspx) moves. The source object typically needs the value of the slider only when the slider's [Value](https://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.rangebase.value(v=vs.110).aspx) stops changing. To prevent updating the source too often, use [Delay](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingbase.delay(v=vs.110).aspx) to specify that the source should not be updated until a certain amount of time elapses after the thumb stops moving.

## Binding to types that Implement ICustomTypeProvider

WPF supports data binding to objects that implement [ICustomTypeProvider](https://msdn.microsoft.com/en-us/library/system.reflection.icustomtypeprovider(v=vs.110).aspx), also known as custom types. You can use custom types in the following cases.

1. As a [PropertyPath](https://msdn.microsoft.com/en-us/library/system.windows.propertypath(v=vs.110).aspx) in a data binding. For example, the [Path](https://msdn.microsoft.com/en-us/library/system.windows.data.binding.path(v=vs.110).aspx) property of a [Binding](https://msdn.microsoft.com/en-us/library/system.windows.data.binding(v=vs.110).aspx) can reference a property of a custom type.
2. As the value of the [DataType](https://msdn.microsoft.com/en-us/library/system.windows.datatemplate.datatype(v=vs.110).aspx) property.
3. As a type that determines the automatically generated columns in a [DataGrid](https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid(v=vs.110).aspx).

## Retrieving data binding information from a binding expression

In certain cases, you might get the [BindingExpression](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpression(v=vs.110).aspx) of a [Binding](https://msdn.microsoft.com/en-us/library/system.windows.data.binding(v=vs.110).aspx) and need information about the source and target objects of the binding. New APIs have been added to enable you to get the source or target object or the associated property. When you have a [BindingExpression](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpression(v=vs.110).aspx), use the following APIs to get information about the target and source.

| **To find this value of the binding** | **Use this API** |
| --- | --- |
| The target object | [BindingExpressionBase.Target](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpressionbase.target(v=vs.110).aspx) |
| The target property | [BindingExpressionBase.TargetProperty](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpressionbase.targetproperty(v=vs.110).aspx) |
| The source object | [BindingExpression.ResolvedSource](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpression.resolvedsource(v=vs.110).aspx) |
| The source property | [BindingExpression.ResolvedSourcePropertyName](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpression.resolvedsourcepropertyname(v=vs.110).aspx) |
| Whether the [BindingExpression](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpression(v=vs.110).aspx) belongs to a [BindingGroup](https://msdn.microsoft.com/en-us/library/system.windows.data.bindinggroup(v=vs.110).aspx) | [BindingExpressionBase.BindingGroup](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingexpressionbase.bindinggroup(v=vs.110).aspx) |
| The owner of a [BindingGroup](https://msdn.microsoft.com/en-us/library/system.windows.data.bindinggroup(v=vs.110).aspx) | [Owner](https://msdn.microsoft.com/en-us/library/system.windows.data.bindinggroup.owner(v=vs.110).aspx) |

## Checking for a valid DataContext object

There are cases where the [DataContext](https://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext(v=vs.110).aspx) of an item container in an [ItemsControl](https://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol(v=vs.110).aspx) becomes disconnected. An item container is the UI element that displays an item in an [ItemsControl](https://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol(v=vs.110).aspx). When an [ItemsControl](https://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol(v=vs.110).aspx) is data bound to a collection, an item container is generated for each item. In some cases, item containers are removed from the visual tree. Two typical cases where an item container is removed are when an item is removed from the underlying collection and when virtualization is enabled on the [ItemsControl](https://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol(v=vs.110).aspx). In these cases, the [DataContext](https://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext(v=vs.110).aspx) property of the item container will be set to the sentinel object that is returned by the [BindingOperations.DisconnectedSource](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingoperations.disconnectedsource(v=vs.110).aspx) static property. You should check whether the [DataContext](https://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext(v=vs.110).aspx) is equal to the [DisconnectedSource](https://msdn.microsoft.com/en-us/library/system.windows.data.bindingoperations.disconnectedsource(v=vs.110).aspx) object before accessing the [DataContext](https://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext(v=vs.110).aspx) of an item container.

## Repositioning data as the data's values change (Live shaping)

A collection of data can be grouped, sorted, or filtered. WPF 4.5 enables the data to be rearranged when the data is modified. For example, suppose that an application uses a [DataGrid](https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid(v=vs.110).aspx) to list stocks in a stock market and the stocks are sorted by stock value. If live sorting is enabled on the stocks' [CollectionView](https://msdn.microsoft.com/en-us/library/system.windows.data.collectionview(v=vs.110).aspx), a stock's position in the [DataGrid](https://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid(v=vs.110).aspx) moves when the value of the stock becomes greater or less than another stock's value. For more information, see the [ICollectionViewLiveShaping](https://msdn.microsoft.com/en-us/library/system.componentmodel.icollectionviewliveshaping(v=vs.110).aspx) interface.

## Improved Support for Establishing a Weak Reference to an Event

Implementing the weak event pattern is now easier because subscribers to events can participate in it without implementing an extra interface. The generic [WeakEventManager](https://msdn.microsoft.com/en-us/library/system.windows.weakeventmanager(v=vs.110).aspx) class also enables subscribers to participate in the weak event pattern if a dedicated [WeakEventManager](https://msdn.microsoft.com/en-us/library/system.windows.weakeventmanager(v=vs.110).aspx) does not exist for a certain event. For more information, see [Weak Event Patterns](https://msdn.microsoft.com/en-us/library/aa970850(v=vs.110).aspx).

## New methods for the Dispatcher class

The Dispatcher class defines new methods for synchronous and asynchronous operations. The synchronous [Invoke](https://msdn.microsoft.com/en-us/library/hh199416(v=vs.110).aspx) method defines overloads that take an [Action](https://msdn.microsoft.com/en-us/library/system.action(v=vs.110).aspx) or [Func<TResult>](https://msdn.microsoft.com/en-us/library/bb534960(v=vs.110).aspx) parameter. The new asynchronous method, [InvokeAsync<TResult>](https://msdn.microsoft.com/en-us/library/hh199385(v=vs.110).aspx), also takes an [Action](https://msdn.microsoft.com/en-us/library/system.action(v=vs.110).aspx) or [Func<TResult>](https://msdn.microsoft.com/en-us/library/bb534960(v=vs.110).aspx) as the callback parameter and returns a [DispatcherOperation](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcheroperation(v=vs.110).aspx) or [DispatcherOperation<TResult>](https://msdn.microsoft.com/en-us/library/hh199415(v=vs.110).aspx). The [DispatcherOperation](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcheroperation(v=vs.110).aspx) and [DispatcherOperation<TResult>](https://msdn.microsoft.com/en-us/library/hh199415(v=vs.110).aspx) classes define a [Task](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx) property. When you call [InvokeAsync](https://msdn.microsoft.com/en-us/library/hh199422(v=vs.110).aspx), you can use the await keyword with either the [DispatcherOperation](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcheroperation(v=vs.110).aspx) or the associated [Task](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx). If you need to wait synchronously for the [Task](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx) that is returned by a [DispatcherOperation](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcheroperation(v=vs.110).aspx) or [DispatcherOperation<TResult>](https://msdn.microsoft.com/en-us/library/hh199415(v=vs.110).aspx), call the [DispatcherOperationWait](https://msdn.microsoft.com/en-us/library/hh199383(v=vs.110).aspx) extension method. Calling [Task.Wait](https://msdn.microsoft.com/en-us/library/dd270644(v=vs.110).aspx) will result in a deadlock if the operation is queued on a calling thread. For more information about using a [Task](https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx) to perform asynchronous operations, see [Task Parallelism (Task Parallel Library)](https://msdn.microsoft.com/en-us/library/dd537609(v=vs.110).aspx).

## Markup Extensions for Events

WPF 4.5 supports markup extensions for events. While WPF does not define a markup extension to be used for events, third parties are able to create a markup extension that can be used with events.

# WPF Dispatcher - Introduction and How to use?

[**http://dotnetpattern.com/wpf-dispatcher**](http://dotnetpattern.com/wpf-dispatcher)

## Thread Apartments

All objects in the process are grouped into Apartments.

There are two types of apartments in Threads:

* Single-Threaded Apartments
* Multi-Threaded Apartments

### Single-Threaded Apartment (STA)

Single-threaded apartments contains only one thread. All objects in this apartment can receive method calls from only this thread. Objects does not need synchronization because all methods calls are comes synchronously from single thread.

Single-threaded apartment needs a message queue to handle calls from other threads. When other threads calls an object in STA thread then the method call are queued in the message queue and STA object will receive a call from that message queue.

### Multi-Threaded Apartment (MTA)

Multi-threaded apartments contains one or more threads. All objects in this apartment can receive calls from any thread. All objects are self responsible for maintaining the synchronization of their data.

## WPF Dispatcher

A WPF application must start in single-threaded apartment thread. STA have a message queue to synchronize method calls within his apartment. As well as other threads outside the apartment can't update the objects directly. They must place their method call into the message queue to update the objects in STA.

***Dispatcher owns the message queue for the STA thread.***

When you execute a WPF application, it automatically create a new Dispatcher object and calls its Run method. Run method is used for initializing the message queue.

When WPF application starts, it creates two threads:

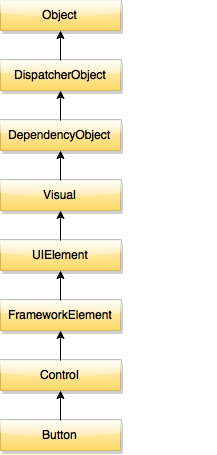
1. Render thread
2. UI thread

UI thread is responsible all the user inputs, handle events, paints screen and run the application code. Render threads runs in the background and used for render the WPF screen.

WPF Dispatcher is associated with the UI thread. The UI thread queues methods call inside the Dispatcher object. Whenever your changes the screen or any event executes, or call a method in the code-behind all this happen in the UI thread and UI thread queue the called method into the Dispatcher queue. Dispatcher execute its message queue into the synchronous order.

### How all WPF objects refers to single Dispatcher?

Every WPF control whether it is Window, button or textbox inherits from DispatcherObject. Below is class hierarchy diagram.



When WPF creates an instance of Button, it calls the protected constructor of DispatcherObject. DispatcherObject contains a property of type Dispatcher. In the constructor, it save the reference of current thread Dispatcher to Dispatcher property of DispatcherObject.

## Why we need Dispatcher?

WPF works with Dispatcher object behind the scenes and we don't need to work with Dispatcher when we are working on the UI thread.

When we create a new thread for offloading the work and want to update the UI from the other thread then we must need Dispatcher. Only Dispatcher can update the objects in the UI from non-UI thread.

Dispatcher provides two methods for registering method to execute into the message queue.

1. Invoke
2. BeginInvoke

### Invoke

Invoke method takes an Action or Delegate and execute the method synchronously. That means it does not return until the Dispatcher complete the execution of the method.

Here is an example of Invoke:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | public partial class MainWindow : Window  {      public MainWindow()      {          InitializeComponent();          Task.Factory.StartNew(() =>              {                  InvokeMethodExample();              });      }        private void InvokeMethodExample()      {          Thread.Sleep(2000);          Dispatcher.Invoke(() =>              {                  btn1.Content = "By Invoke";              });      }  } |

Above code will create a new thread using Task.Factory and immediately start the thread. In the InvokeMethodExample if we try to directly call to update the Content property of btn1 object. It will throws a System.InvalidOperationException. We have used Invoke method of Dispatcher. In the Invoke method, I pass the Action and update the Content property of Button object. It will not throws any error and successfully update the Content property.

### BeginInvoke

BeginInvoke method take a Delegate but it executes the method asynchronously. That means it immediately returns before calling the method.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public MainWindow()  {      InitializeComponent();      Task.Factory.StartNew(() =>          {              BeginInvokeExample();          });  }    private void BeginInvokeExample()  {      DispatcherOperation op = Dispatcher.BeginInvoke((Action)(() => {              btn1.Content = "By BeginInvoke";          }));  } |

BeginInvoke returns a DispatcherOperation object. This object can be used for knowing the status of operation whether it is completed or not. It also provides two event Aborted and Completed.

# Simple Validation in WPF



[**Gerald Gomes**](https://www.codeproject.com/script/Membership/View.aspx?mid=9690530), 2 Dec 2013 [CPOL](http://www.codeproject.com/info/cpol10.aspx)

|  |  |
| --- | --- |
| https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-fill-md.png  https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-empty-md.png | 4.31 (26 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](https://www.codeproject.com/Tips/690130/Simple-Validation-in-WPF) |  | |

A very simple example of displaying validation error next to controls in WPF

## Introduction

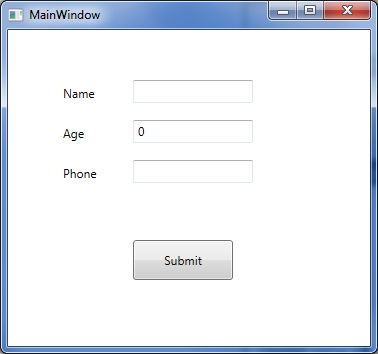
This is a simple example of validation in XAML for WPF controls and displaying error messages.

## Background

I was looking for something out-of-the-box from WPF where no extra coding of style or template is needed for displaying validation errors, where we just need to code the validation logic for each control and it should automatically display an error icon or message next to the control. However, I did not find anything straightforward like that in WPF. But it can be achieved in two simple steps.

## Using the Code

Here is a very simple form in XAML that is created which has three textbox controls:



Let us add code so that when values are entered in the above text boxes, they automatically run validation and if there are validation errors, they will be displayed next to the corresponding control. In order to do this, the following two steps are needed:

1. Create a ControlTemplate with AdornedElementPlaceHolder
2. Implement a validation class inheriting the abstract class called ValidationRule

Here is the sample validation control template. Let us start with a very simple validation control template where all we have is a TextBlock which will display a red exclamatory sign next to the control that has an error.

Hide   Copy Code

<ControlTemplate x:Key="validationErrorTemplate">

<DockPanel>

**<TextBlock Foreground="Red"**

**DockPanel.Dock="Top">!</TextBlock>**

<AdornedElementPlaceholder

x:Name="ErrorAdorner"

></AdornedElementPlaceholder>

</DockPanel>

</ControlTemplate>

Now, let us also create a validator class by inheriting from the ValidationRule class and implementing its abstract method as below:

Hide   Copy Code

public class NameValidator : ValidationRule

{

public override ValidationResult Validate

(object value, System.Globalization.CultureInfo cultureInfo)

{

if (value == null)

return new ValidationResult(false, "value cannot be empty.");

else

{

if (value.ToString().Length > 3)

return new ValidationResult

(false, "Name cannot be more than 3 characters long.");

}

return ValidationResult.ValidResult;

}

}

Let's plug this validation control template and the validation rule with control that we want to validate.

Hide   Copy Code

<TextBox Height="23" HorizontalAlignment="Left"

Grid.Column="1" Grid.Row="0" Name="textBox1"

VerticalAlignment="Top" Width="120"

Validation.ErrorTemplate="{StaticResource validationErrorTemplate}"

>

<TextBox.Text>

<Binding Path="Name" Mode="TwoWay"

UpdateSourceTrigger="LostFocus">

<Binding.ValidationRules>

<local:NameValidator></local:NameValidator>

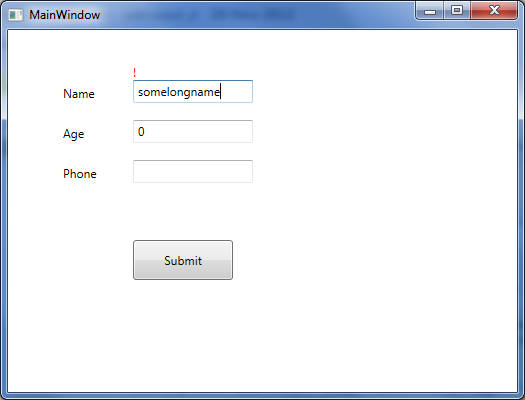
</Binding.ValidationRules>

</Binding>

</TextBox.Text>

</TextBox>

When we run this now and enter a name longer than three characters long, it displays the red exclamatory sign indicating validation error.



Now let us just replace the TextBlock in the above validation control template code (the line that is in bold) with the StackPanel containing an ellipse and a TextBlock to display the same validation error, as below:

Hide   Copy Code

<ControlTemplate x:Key="validationErrorTemplate">

<DockPanel>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">

<Grid Width="12" Height="12">

<Ellipse Width="12" Height="12"

Fill="Red" HorizontalAlignment="Center"

VerticalAlignment="Center"

></Ellipse>

<TextBlock Foreground="White" FontWeight="Heavy"

FontSize="8" HorizontalAlignment="Center"

VerticalAlignment="Center" TextAlignment="Center"

ToolTip="{Binding ElementName=ErrorAdorner,

Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"

>X</TextBlock>

</Grid>

<TextBlock Foreground="Red" FontWeight="12" Margin="2,0,0,0"

Text="{Binding ElementName=ErrorAdorner,

Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"

></TextBlock>

</StackPanel>

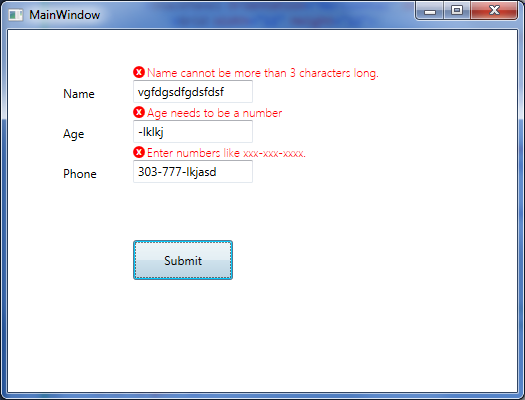
<AdornedElementPlaceholder

x:Name="ErrorAdorner" ></AdornedElementPlaceholder>

</DockPanel>

</ControlTemplate>

Now when we run the code and validation fails, a validation error will be displayedas shown in the screenshot below (coded validator class for age and phone number as well).



That is all that is needed for the simplest validation error to show up next to the control. Notice in the validation control template, we are using a DockPanel as the layout control, therefore we can easily change where the error icon and error message will be displayed. We can display them on top of the control that is failing validation (as the above picture), or on the left, right, or bottom.

[**https://www.codeproject.com/Tips/690130/Simple-Validation-in-WPF**](https://www.codeproject.com/Tips/690130/Simple-Validation-in-WPF)

# Data Validation in WPF Using IDataErrorInfo and Displaying the Error in a ToolTip

* [](http://www.c-sharpcorner.com/members/princy-gupta)

* [Princy Gupta](http://www.c-sharpcorner.com/members/princy-gupta)

* Apr 16 2014

* [Article](http://www.c-sharpcorner.com/articles)
* [3](javascript:void(0);)
* [2](javascript:void(0);)
* 91.4k
* [facebook](https://www.facebook.com/sharer.php?u=http://goo.gl/XPzcFG)
* [twitter](https://twitter.com/intent/tweet?&via=CsharpCorner&related=CsharpCorner&text=Data+Validation+in+WPF+Using+IDataErrorInfo+and+Displaying+the+Error+in+a+ToolTip%20http://goo.gl/XPzcFG)
* [linkedIn](https://www.linkedin.com/shareArticle?mini=true&url=http://goo.gl/XPzcFG)
* [google Plus](https://plus.google.com/share?url=http://goo.gl/XPzcFG)
* [Reddit](http://www.reddit.com/submit?url=http%3a%2f%2fwww.c-sharpcorner.com%2fUploadFile%2fprincy.scorpin%2fdata-validation-in-wpf-using-idataerrorinfo-and-displaying-t%2fdefault.aspx)
* [Expand](javascript:void(0);)

[Download 100% FREE Spire Office APIs](http://www.c-sharpcorner.com/AdRedirector.aspx?AdId=92&target=http://www.e-iceblue.com/Introduce/free-products.html?aff_id=100)

**Introduction**  
This article explains how data validation works in WPF. When user enters some data, to validate that data the .NET Framework has the interface IDataErrorInfo Interface. Using this interface the user input validation becomes automatic and very easy to display.  
   
**Sample Code to implement IDataErrorInfo**

In the following example, when the user inputs an invalid data, it displays a red coloured border around the TextBox and when a user hovers the cursor on the invalidated control, a ToolTip appears with a message describing the problem.

1. Create a new WPF application.
2. Add two textboxs and a button to the MainWindow.xaml.
3. Add a ViewModel for this MainWindow.xaml. Create a class file and name it as MainWindowViewModel.cs. Now we need to bind this class to the WPF window.

**View**

Set the ViewModel “MainWindowViewModel” as the DataContext.

public partial class MainWindow : Window

{

    public MainWindow()

    {

        InitializeComponent();

        DataContext = new MainWindowViewModel();

    }

}

**ViewModel**  
  
In the ViewModel add a string property “ValidateInputText” and an int property “Age” to bind to the textboxs and an ICommand to bind to the button used for validation.

using System;

using System.Windows.Input;

using System.ComponentModel;

namespace WpfApplication1

{

    public class MainWindowViewModel : IDataErrorInfo

    {

        public MainWindowViewModel()

        {

        }

        //property to bind to textbox

        public string ValidateInputText

        {

            get;

            set;

        }

        //ICommand to bind to button

        public ICommand ValidateInputCommand

        {

            get { return new RelayCommand(); }

            set { }

        }

        private int age = 20;

        public int Age

        {

            get { return age; }

            set { age = value; }

        }

        #region IDataErrorInfo

        //In this region we are implementing the properties defined in //the IDataErrorInfo interface in System.ComponentModel

        public string this[string columnName]

        {

            get

            {

                if ("ValidateInputText" == columnName)

                {

                    if (String.IsNullOrEmpty(ValidateInputText))

                    {

                        return "Please enter a Name";

                    }

                }

                else if ("Age" == columnName)

                {

                    if (Age < 0)

                    {

                        return "age should be greater than 0";

                    }

                }

                return "";

            }

        }

        public string Error

        {

            get { throw new NotImplementedException(); }

        }

    }

        #endregion

    class RelayCommand : ICommand

    {

        public void Execute(object parameter)

        {

        }

        public bool CanExecute(object parameter)

        {

            return true;

        }

        public event EventHandler CanExecuteChanged;

    }

}

**UI**

<window x:class="WpfApplication1.MainWindow"xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" title="MainWindow" height="126"

    width="525">

  <Window.Resources>

    <!—Error Template to change the default behaviour-->

    <ControlTemplate x:Key="ErrorTemplate">

      <DockPanel LastChildFill="True">

        <Border BorderBrush="Red" BorderThickness="1">

          <AdornedElementPlaceholder />

        </Border>

      </DockPanel>

    </ControlTemplate>

    <!—To display tooltip with the error-->

    <Style TargetType="TextBox">

      <Style.Triggers>

        <Trigger Property="Validation.HasError" Value="true">

          <Setter Property="ToolTip"

          Value="{Binding RelativeSource={x:Static RelativeSource.Self},

Path=(Validation.Errors)[0].ErrorContent}"/>

        </Trigger>

      </Style.Triggers>

    </Style>

  </Window.Resources>

  <!—TextBox and button Binding-->

  <Grid>

    <TextBox Height="23" Name="textBox1" Margin="107,24,0,116" HorizontalAlignment="Left" Width="236"

    Text="{Binding ValidateInputText, Mode=TwoWay, ValidatesOnDataErrors=True, NotifyOnValidationError=True, ValidatesOnExceptions=True}"

    Validation.ErrorTemplate="{StaticResource ErrorTemplate}"/>

    <TextBox Height="23" Name="textBox2" Margin="107,53,0,87" HorizontalAlignment="Left" Width="236"

    Text="{Binding Age, Mode=TwoWay, ValidatesOnDataErrors=True, NotifyOnValidationError=True, ValidatesOnExceptions=True}"

    Validation.ErrorTemplate="{StaticResource ErrorTemplate}"/>

    <Button Content="Validate" Height="23" HorizontalAlignment="Left" Margin="107,82,0,0" Name="button1"

    Command="{Binding ValidateInputCommand}" VerticalAlignment="Top" Width="75" />

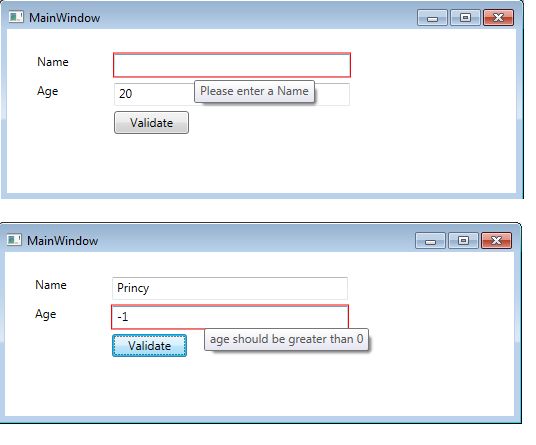
    <Label Content="Name" Height="28" HorizontalAlignment="Left" Margin="25,19,0,0" Name="label1"VerticalAlignment="Top" />

    <Label Content="Age" Height="28" HorizontalAlignment="Left" Margin="25,48,0,0" Name="label2"VerticalAlignment="Top" />

  </Grid>

</window>

**Final Output**

  
**Explanation**

1. Once you have created the View and the ViewModel and the basic binding is complete, we now need to implement the IDataErrorInfo. The sample code for that is provided above. The User would need to inherit from the IDataErrorInfo interface found in the System.ComponentModel namespace. We would then need to implement it in the ViewModel.
2. In order to enable validation, you need to add ValidatesOnDataErrors=True, NotifyOnValidationError=True and ValidatesOnExceptions=True in your view for the textboxes.
3. Now run the application, you will observe the default red colour border around the TextBox, you can change the default behaviour by applying an ErrorTemplate. This template is applied to the “Validation.ErrorTemplate” property.
4. In order to display the tooltip you would need to add a trigger to the TextBox control for its “Validation.HasError” property. This property gets the value of any error on the bound control.

[**http://www.c-sharpcorner.com/UploadFile/princy.scorpin/data-validation-in-wpf-using-idataerrorinfo-and-displaying-t/**](http://www.c-sharpcorner.com/UploadFile/princy.scorpin/data-validation-in-wpf-using-idataerrorinfo-and-displaying-t/)