

# Unit - VI

## Overview

- The Windows Presentation Foundation is Microsoft's next generation UI framework to create applications with a rich user experience.
- It is part of the .NET framework 3.0 and higher.
- WPF combines application UIs, 2D graphics, 3D graphics, documents and multimedia into one single framework. Its vector based rendering engine uses hardware acceleration of modern graphic cards. This makes the UI faster, scalable and resolution independent.
- The following illustration gives you an overview of the main new features of WPF.



### **Advantages of WPF over a Winform**

#### **A : Any where execution**

WPF use xaml(extensive application markup language) for designing . Its easy to implement any where.

#### **B: Binding**

Binding of two textbox or any other control can easily have implemented without coding.

#### **C: Common Look and Feel**

For eg. If user want to set background color of textbox control, then in simple windows application user has to change background color of individual textbox. But in WPF its possible style property of control.

#### **D: Directive Programming**

No need to write a much code for designing.

#### **E: Expression blend and Animation**

Due to of Directx is easy to animation

## Unit - VI

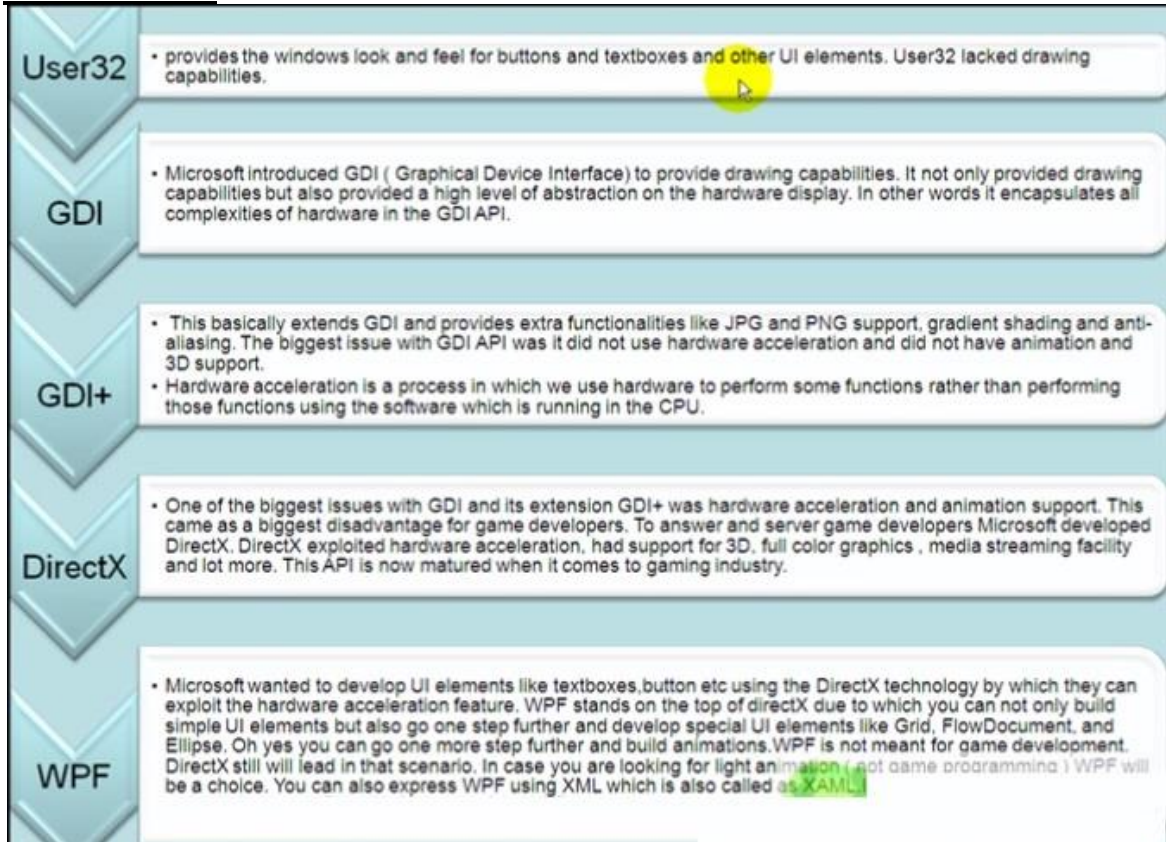
### F: Faster Execution

WPF using Hardware rendering so execution of WPF application are more faster then winforms.

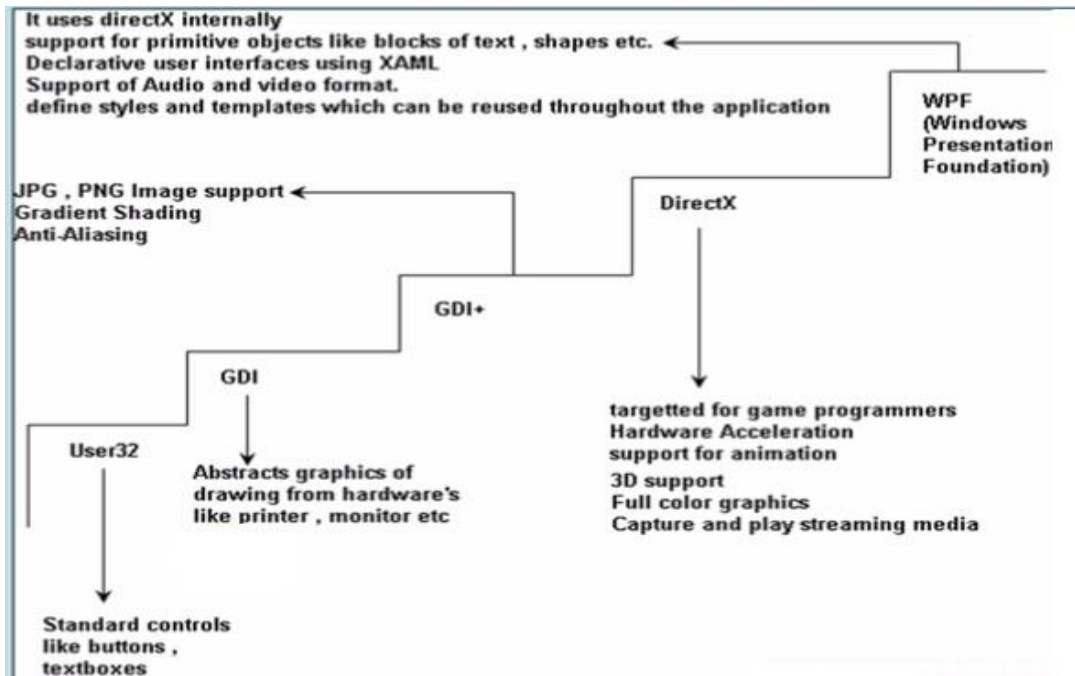
### G: Graphical Independency

In WPF application no need to set resolution for application it is automatic set resolution as per device.

### Need of WPF



## Unit - VI



### Features & Benefits:

#### Separation of Appearance and Behavior

WPF separates the appearance of a user interface from its behavior. The appearance is generally specified in the Extensible Application Markup Language (XAML), the behavior is implemented in a managed programming language like C# or Visual Basic. The two parts are tied together by data binding, events and commands. The separation of appearance and behavior brings the following benefits:

- Appearance and behavior are loosely coupled
- Designers and developers can work on separate models.
- Graphical design tools can work on simple XML documents instead of parsing code.

#### Rich composition

Controls in WPF are extremely composable. You can define almost any type of controls as content of another. Although this flexibility sounds horrible to designers, it's a very powerful feature if you use it appropriately. Put an image into a button to create an image button, or put a list of videos into a combobox to choose a video file.



## Unit - VI

```
<Button>
  <StackPanel Orientation="Horizontal">
    <Image Source="speaker.png" Stretch="Uniform"/>
    <TextBlock Text="Play Sound" />
  </StackPanel>
</Button>
```

### Highly customizable

Because of the strict separation of appearance and behavior you can easily change the look of a control. The concept of styles let you skin controls almost like CSS in HTML. Templates let you replace the entire appearance of a control.

The following example shows an default WPF button and a customized button.



### Resolution independence

All measures in WPF are logical units - not pixels. A logical unit is a 1/96 of an inch. If you increase the resolution of your screen, the user interface stays the same size - it just gets crispier.

Since WPF builds on a vector based rendering engine it's incredibly easy to build scalable user interfaces.

## Unit - VI

### **Type of Application**

.NET Framework, System.Window, and mark-up and code-behind, constitute of the WPF application development experience.

Additionally, WPF has comprehensive features for creating user experience with rich content.

To package this contents and deliver it to user as “applications”. WPF provide types and services that are collectively known as the application model. The application model supports the development of both standalone and browser-hosted applications.

1. **Standalone Application:** For standalone applications, you can use the Window class to create windows and dialog boxes that are accessed from menu bar and tool bars.
2. **Browser- Hosted Applications:** For browser hosted applications known as XAML browser application (XBAPs), you can create pages that you can navigate between using hyperlinks.
3. **Custom Controls Libraries:** (non-executable assemblies containing reusable controls).

# Unit - VI

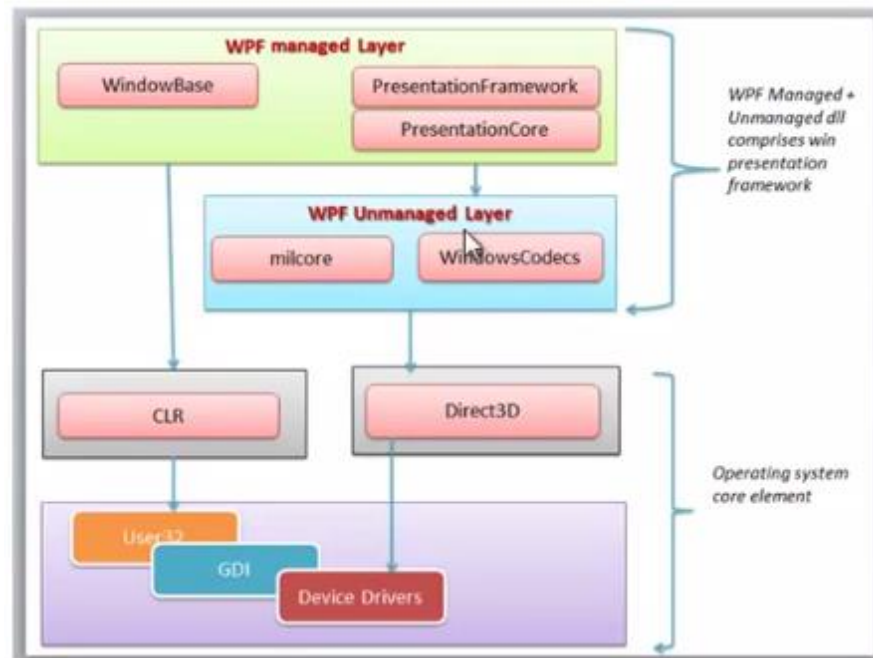
## WPF Architecture

WPF uses a multilayered architecture. At the top, your application interacts with a high-level set of services that are completely written in managed C# code.

Managed Code is what Visual Basic .NET and C# compilers create. It compiles to Intermediate Language (IL), not to machine code that could run directly on your computer. Managed code runs in the Common Language Runtime.

The actual work of translating .NET objects into Direct3D textures and triangles happens behind the scenes, using a lower level unmanaged component called milcore.dll.

Unmanaged code is what you use to make before Visual Studio .NET 2002 was released. Visual Basic 6, Visual C++ 6, heck, even that 15-year old C compiler you may still have kicking around on your hard drive all produced unmanaged code.



The following are different layer in WPF Architecture

- Managed Layer
- Unmanaged Layer
- Core API

**Managed Layer:** Managed layer of WPF is built using a number of assemblies. These assemblies build up the WPF framework, communicates with lower level unmanaged API to render its content. The few assemblies that comprise the WPF framework are:

## Unit - VI

- **PresentationFramework.dll:** Creates the top level elements like layout panel, controls, windows, styles etc.
- **PresentationCore.dll:** It holds base types such as UI element, Visual from which all shapes and controls are derived in PresentationFramework.dll.
- **WindowBase.dll:** They hold even more basic element which are capable to used outside the WPF environment like Dispatcher object, Dependency Object.

**Unmanaged Layer (milcore.dll):** The unmanaged layer of WPF is called milcore or Media Integration Library Core. It is basically translating the WPF higher level objects like panels, buttons, animation etc, into textures that Direct3D expects. It is the main rendering engine in WPF.

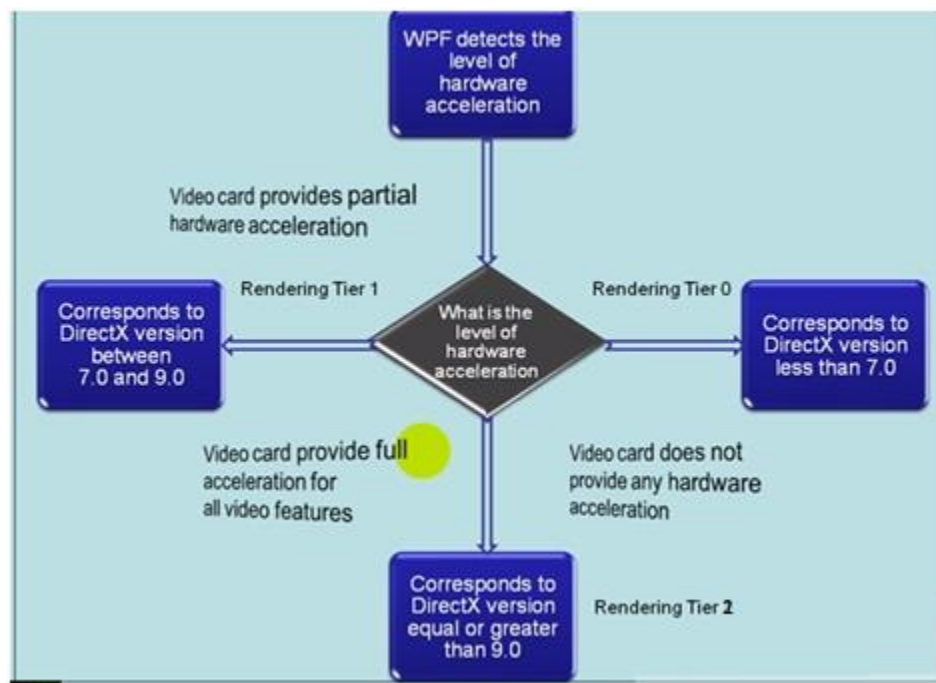
**WindowsCodecs.dll:** This is another low level API which is used for imaging support in WPF applications. WindowsCodecs.dll comprises of a number of codes which encodes/decodes images into vector graphics that would be rendered into WPF screen.

**Direct3D:** It is the low level API in which the graphics of WPF is rendered.

**GDI & Device Driver:** GDI and Device Drivers are specific to the OS which is also used from application to access low level APIs.

**User32:** It is the primary core API which every program uses, It actually manages memory and process separation.

### Three tier Rendering



## Unit - VI

### Version History of WPF

**WPF 3.0:** The first version of WPF was released.

**WPF 3.5:** A year later, a new version of WPF was released as part of the .NET framework 3.5. The new features in WPF are mostly minor refinements, including bug fixes and performance improvements.

**WPF3.5 SP1:** When the .NET framework SP1 was released, the designers of WPF had a chance to slip in a few new features, such as slick graphical effects and the sophisticated DataGrid control.

**WPF 4:** WPF adds a number of refinements, including some valuable new features that build on the existing WPF infrastructure. Some of the notable changes include better text rendering, more natural animation, and support for Window 7 features such as multitouch and the new taskbar.

## DEVELOP FIRST APPLICATION IN WPF



# Unit - VI

## Layout in WPF








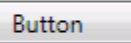







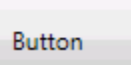
Layout of controls is critical to an applications usability. Arranging controls based on fixed pixel coordinates may work for an limited enviroment, but as soon as you want to use it on different screen resolutions or with different font sizes it will fail. WPF provides a rich set built-in layout panels that help you to avoid the common pitfalls.

These are the five most popular layout panels of WPF:

- Grid Panel
- Stack Panel
- Dock Panel
- Wrap Panel
- Canvas Panel

## Vertical and Horizontal Alignment

Use the VerticalAlignment and HorizontalAlignmant properties to dock the controls to one or multiple sides of the panel. The following illustrations show how the sizing behaves with the different combinations.

		HorizontalAlignment			
		Left	Center	Right	Strech
VerticalAlignment	Top				
	Center				
	Bottom				
	Stretch				

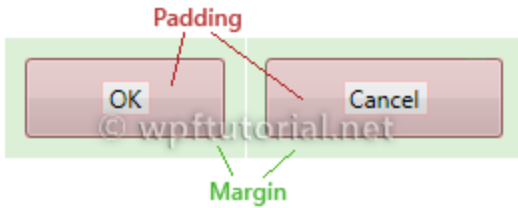
## Margin and Padding

The Margin and Padding properties can be used to reserve some space around of within the control.

- The Margin is the extra space **around** the control.
- The Padding is extra space **inside** the control.

## Unit - VI

- The Padding of an outer control is the Margin of an inner control.



### Height and Width

Although it's not a recommended way, all controls provide a Height and Width property to give an element a fixed size. A better way is to use the MinHeight, MaxHeight, MinWidth and MaxWidth properties to define an acceptable range. If you set the width or height to Auto the control sizes itself to the size of the content.

### Stack Panel

The StackPanel in WPF is a simple and useful layout panel. It stacks its child elements below or beside each other, depending on its orientation. This is very useful to create any kinds of lists. All WPF ItemsControls like ComboBox, ListBox or Menu use a StackPanel as their internal layout panel.

```
<StackPanel>
  <TextBlock Margin="10" FontSize="20">How do you like your coffee?</TextBlock>
  <Button Margin="10">Black</Button>
  <Button Margin="10">With milk</Button>
  <Button Margin="10">Latte machiato</Button>
  <Button Margin="10">Chappuchino</Button>
</StackPanel>
```

### Grid

The grid is a layout panel that arranges its child controls in a tabular structure of rows and columns. Its functionality is similar to the HTML table but more flexible. A cell can contain multiple controls, they can span over multiple cells and even overlap themselves.

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

## Unit - VI

```
<ColumnDefinition Width="200" />
</Grid.ColumnDefinitions>
<Label Grid.Row="0" Grid.Column="0" Content="Name:"/>
<Label Grid.Row="1" Grid.Column="0" Content="E-Mail:"/>
<Label Grid.Row="2" Grid.Column="0" Content="Comment:"/>
<TextBox Grid.Column="1" Grid.Row="0" Margin="3" />
<TextBox Grid.Column="1" Grid.Row="1" Margin="3" />
<TextBox Grid.Column="1" Grid.Row="2" Margin="3" />
<Button Grid.Column="1" Grid.Row="3" HorizontalAlignment="Right"
        MinWidth="80" Margin="3" Content="Send" />
</Grid>
```

# Unit - VI

## Controls

### TextBox:

```
<TextBox SpellCheck.IsEnabled="True" Language="en-US" />
```

### Expander:

```
<Expander Header="More Options">
    <StackPanel Margin="10,4,0,0">
        <CheckBox Margin="4" Content="Option 1" />
        <CheckBox Margin="4" Content="Option 2" />
        <CheckBox Margin="4" Content="Option 3" />
    </StackPanel>
</Expander>
```

### PasswordBox

```
<PasswordBox x:Name="passwordBox" PasswordChar="*" />
```

### RadioButton

```
<StackPanel>
    <RadioButton GroupName="Os" Content="Windows XP" IsChecked="True"/>
    <RadioButton GroupName="Os" Content="Windows Vista" />
    <RadioButton GroupName="Os" Content="Windows 7" />
    <RadioButton GroupName="Office" Content="Microsoft Office 2007"
IsChecked="True"/>
    <RadioButton GroupName="Office" Content="Microsoft Office 2003"/>
    <RadioButton GroupName="Office" Content="Open Office"/>
</StackPanel>
```

### Slider:

```
<Slider Minimum="0"
Maximum="20"
IsSnapToTickEnabled="True"
TickFrequency="2">
```

### TextBlock:

```
<TextBlock Text="This is a
multiline text."
LineHeight="25">
```

### SpeechSynthesizer:

```
SpeechSynthesizer synthesizer = new SpeechSynthesizer();
    synthesizer.Volume = 100; // 0...100
    synthesizer.Rate = 1; // -10...10

    // Synchronous
    synthesizer.Speak("Welcome to the WPF Workshop. WPF is very easy, it is called 2nd generation
of UI Framework ");
    // Asynchronous
    synthesizer.SpeakAsync("Welcome to CMPICA.");
```

# Unit - VI

## Animations

Animation in WPF has been made easier because WPF achieves animation by modifying properties of elements, whereas in Windows Forms, a developer has to create a timer and modify the appearance of elements on the tick event of a timer. WPF uses its own timing system which can be written using managed code and XAML. The internal work of redrawing the screen is handled efficiently by WPF. While animating using WPF, you just need to focus on the effects you want to create without bothering about how to achieve those effects.

### DoubleAnimation

WPF achieves animation by animating element properties. For example, if you want to produce a zoom in or zoom out effect for a rectangle, you can animate the width and height properties. The following code animates a rectangle by modifying its width and height properties.

```
<Rectangle Name="myrect" Width="1" Height="1">
  <Rectangle.Fill>
    <SolidColorBrush Color="Red"/>
  </Rectangle.Fill>
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Window.Loaded">
      <BeginStoryboard>
        <Storyboard RepeatBehavior="Forever">
          <DoubleAnimation Storyboard.TargetName="myrect"
            Storyboard.TargetProperty="Width" From="1" To="350"
            Duration="0:0:1" BeginTime="0:0:0"/>
          <DoubleAnimation Storyboard.TargetName="myrect"
            Storyboard.TargetProperty="Height" From="1" To="250"
            Duration="0:0:1" BeginTime="0:0:1"/>
          <DoubleAnimation Storyboard.TargetName="myrect"
            Storyboard.TargetProperty="Height" From="250"
            To="1" Duration="0:0:1" BeginTime="0:0:2"/>
          <DoubleAnimation Storyboard.TargetName="myrect"
            Storyboard.TargetProperty="Width" From="350" To="1"
            Duration="0:0:1" BeginTime="0:0:3"/>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

## Unit - VI

This code triggers the animation automatically when the window is loaded. The code adds an EventTrigger to the rectangle. The BeginStoryboard action runs a storyboard. This storyboard uses four DoubleAnimations. The first DoubleAnimation increases the width of the rectangle from 1 to 350. The second one increases the height from 1 to 250. The third and fourth do the reverse by decreasing the height and width back to 1. The four DoubleAnimations are made to run in a sequence by setting the BeginTime attribute such that each animation starts when the previous is over. The RepeatBehavior attribute of the Storyboard is assigned the value "Forever" which makes the animation run indefinitely.

### Fade In and Fade Out Animation Using DoubleAnimation

Fade In and Fade Out Animation effects can be created using the Opacity property as follows:

```
<Rectangle Name="myrect" Width="350" Height="250">
  <Rectangle.Fill>
    <SolidColorBrush x:Name="brush" Color="Red"/>
  </Rectangle.Fill>
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Window.Loaded">
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation Storyboard.TargetName="myrect"
            Storyboard.TargetProperty="Opacity" From="0" To="1"
            Duration="0:0:1" BeginTime="0:0:0" AutoReverse="True"
            RepeatBehavior="Forever"/>
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

### ColorAnimation

We can use ColorAnimation to animate the Color property of a rectangle. Following is the code to produce color animation:

```
<Rectangle Name="myrect" Width="350" Height="250">
  <Rectangle.Fill>
    <SolidColorBrush x:Name="brush" Color="Red"/>
  </Rectangle.Fill>
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Window.Loaded">
```

## Unit - VI

```
<BeginStoryboard>
  <Storyboard RepeatBehavior="Forever">
    <ColorAnimation Storyboard.TargetName="brush"
      Storyboard.TargetProperty="Color" From="Red" To="Green"
      Duration="0:0:1" BeginTime="0:0:0"/>
    <ColorAnimation Storyboard.TargetName="brush"
      Storyboard.TargetProperty="Color" From="Green" To="Blue"
      Duration="0:0:1" BeginTime="0:0:1"/>
    <ColorAnimation Storyboard.TargetName="brush"
      Storyboard.TargetProperty="Color" From="Blue" To="Yellow"
      Duration="0:0:1" BeginTime="0:0:2"/>
    <ColorAnimation Storyboard.TargetName="brush"
      Storyboard.TargetProperty="Color" From="Yellow"
      To="Red" Duration="0:0:1" BeginTime="0:0:3"/>
  </Storyboard>
</BeginStoryboard>
</EventTrigger>
</Rectangle.Triggers>
</Rectangle>
```

### Wheel Animation:

```
DoubleAnimation da = new DoubleAnimation(0,360,new
Duration(TimeSpan.FromSeconds(3)));
da.RepeatBehavior = RepeatBehavior.Forever;

RotateTransform rt = new RotateTransform();
rt.BeginAnimation(RotateTransform.AngleProperty, da);

image1.RenderTransform = rt;
image1.RenderTransformOrigin = new Point(0.5, 0.5);
```

# Unit - VI

## Database Connectivity in WPF:

```
<DataGrid Name="grdEmployee" AutoGenerateColumns="False" Height="350" Width="400"
AlternatingRowBackground="LemonChiffon">
    <DataGrid.Columns>
        <DataGridTextColumn Binding="{Binding EmpName}" Header="Name"></DataGridTextColumn>

    </DataGrid.Columns>
</DataGrid>

SqlConnection con = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\Users\Abhi\Documents\EmpDB.mdf;Integrated Security=True;Connect
Timeout=30;User Instance=True");
SqlDataAdapter da = new SqlDataAdapter("Select * from EmpTable", con);
DataSet ds = new DataSet();
da.Fill(ds);
grdEmployee.ItemsSource = ds.Tables[0].DefaultView;
```

## Consuming Web Service in WPF:

### Web Service

```
public class Service1 : System.Web.Services.WebService
{
    public class WebContent
    {
        public string greet;

        public string[] Names;
    }

    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World";
    }
    [WebMethod]
    public WebContent CreateObject()
    {
        WebContent wc = new WebContent();
        wc.greet = "Good Day";
        wc.Names = new string[5] { "Abhi", "Ankur", "Aakash", "Ajay", "Arpit" };
        return wc;
    }
}

localhost.Service1SoapClient proxy = new localhost.Service1SoapClient();
this.DataContext= proxy.CreateObject();

<ListBox ItemsSource="{Binding Names}"></ListBox>
    <Label VerticalAlignment="Center" HorizontalAlignment="Center" Content="{Binding greet}"></Label>
```