



WINDOWS PRESENTATION FOUNDATION

Windows Presentation Foundation

Lab Book



Document Revision History

Date	Revision No.	Author	Summary of Changes
21-Oct-09	1.0	Ganesh Desai	Initial Document
21-Oct-09	1.0	Ummeaiman Diwanji	Quality Review, Transfer to new template.
15-July-2011	2.0	Ganesh Desai	Changes made as per integration process



Table of Contents

<i>Getting Started</i>	4
<i>Overview</i>	4
<i>Setup Checklist for WPF</i>	4
<i>Lab 1. WPF – Working With XAML</i>	5
1.1: <i>Do As Instructed</i>	5
1.2: <i>Generate Button</i>	6
1.3: <i>Use Properties as Attributes</i>	6
1.4: <i>Properties as Elements</i>	7
1.5: <i>Use Style For Complex Property Values</i>	7
1.6: <i>Using Attached Properties</i>	7
1.7: <i>Using Property Element Syntax</i>	8
1.8: <i>Handling Events of the Controls</i>	9
1.9: <i>Using Markup Extensions</i>	10
1.10: <i>Working With Shapes</i>	11
1.12: <i>Working with Controls</i>	12
1.13: <i>Working with Layouts in XAML</i>	12
1.14: <i>Working With Styles</i>	16
1.15: <i>Working With Resources</i>	19
1.16: <i>Working With Triggers</i>	20
<i>Lab Summary:</i>	21
<i>Lab 2. Develop Rich User Interface With XAML</i>	33
<i>Exercise 2.1: Design GUI</i>	33
<i>Exercise 2.2: Design a Window That Uses Event Trigger Features</i>	35
<i>Appendices</i>	37
<i>Appendix A: Table of Examples</i>	37



Getting Started

Overview

This Lab book is a guided tour for Windows Presentation Foundation (WPF). It contains solved examples and “To Do” assignments. Follow the steps provided in the solved examples and then work out the ‘To Do’ assignments given.

Setup Checklist for WPF

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- *Hardware:* Networked PCs with minimum 1 GB RAM and 60 MB HDD.
Software: Microsoft Visual C# 2012 Express Edition or Visual Studio.NET 2012



Lab 1. WPF – Working With XAML

Goals	<ul style="list-style-type: none">• At the end of this lab session, you will be able to understand the following XAML features:<ul style="list-style-type: none">○ XAML Syntax.○ Properties and their usage.○ Markup Extensions○ Shapes○ WPF Content○ WPF Layouts○ Styles○ Resources○ Triggers
Time	4 Hrs

In this exercise, you will try out various new features available in WPF. You will explore XAML and its features.

1.1: Do As Instructed

Problem Statement:

Setup to try out various WPF features.

Solution:

Step 1: Add a console application in VS2008.

Step 2: Add references to the assemblies: WindowsBase, PresentationCore.

Step 3: Add the following code in the main method:

```
Window mainWindow = new Window();
mainWindow.Title = "WPF Application";
Button button1 = new Button();
button1.Content = "Click Me!";
mainWindow.Content = button1;
Application app = new Application();
app.Run(mainWindow);
```

Example 1: Do As Instructed

Step 4: Run the application.



1.2: Generate Button

Problem Statement:

Generate a button.

Solution:

Step 1: Start a new WPF application in VS2008.

Step 2: Edit the XAML window as follows:

```
<button>Click Me!</button>
```

Example 2: Generate Button

Step 3: Run the application and see how the button is generated.

Note: Any object can declare a default content property. "Click Me!" here is content. There is always a .NET class behind a XAML element. With attributes and child elements, you set the value of properties and define handler methods for events.

1.3: Use Properties as Attributes

Problem Statement:

Use properties as attributes.

Solution:

Step 1: Modify the above code as:

```
<Button Content="Click Me!" Background="LightGreen"/ >
```

Example 3: Use Properties as Attributes

Note: Here, properties are defined as attributes.



1.4: Properties as Elements

Problem Statement:

Instead of using XML attributes, the properties can also be specified as child elements. The value for the content can directly set by specifying the child elements of the Button element. For all other properties of the Button, the name of the child element is defined by the name of the outer element, followed by the property name.

Solution:

Step 1: Edit the above code as:

```
<Button>
  <Button.Background>
    LightGreen
  </Button.Background>
  Click Me!
</Button>
```

Example 4: Properties as Elements

1.5: Use Style For Complex Property Values

Problem Statement:

Use this style when you set complex values for properties.

Solution:

Step 1: Try the following code in XAML:

```
<Button>
  <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Orange" Offset="0.25" />
      <GradientStop Color="Red" Offset="0.75" />
      <GradientStop Color="Violet" Offset="1.0" />
    </LinearGradientBrush>
  </Button.Background>
  Click Me!
</Button>
```

Example 5: Style To Set Complex Property Values

1.6: Using Attached Properties

Problem Statement:



A WPF element can also get features from the parent element. For example, if the Button element is located inside a Canvas element, the button has Top and Left properties that are prefixed with the parent element's name. Such a property is known as attached property.

Solution:

Step 1: Try the following example of attached property:

```
<Canvas>
  <Button Canvas.Top="30" Canvas.Left="40">
    Click Me!
  </Button>
</Canvas>
```

Example 6: Attached Properties

1.7: Using Property Element Syntax

Problem Statement:

Use property element syntax.

Solution:

Step 1: Type the following XAML code:

```
<Ellipse Width="200" Height="150">
  <Ellipse.Fill>
    <LinearGradientBrush>
      <GradientStop Offset="0" Color="Teal"/>
      <GradientStop Offset="1" Color="Aqua"/>
    </LinearGradientBrush>
  </Ellipse.Fill>
</Ellipse>
```

Example 7: Property Element Syntax



Note: We have an *Ellipse* whose *Height* and *Width* properties are being set. The child element of *Ellipse*, on the other hand, is something different than what we have previously seen. In this case, we represent a linear gradient brush to fill the *Ellipse*. But how can we represent all of that information using the standard attribute syntax? It turns out that this is a very common scenario, and to solve the problem we have something called *Property Element Syntax*. In XAML, you can turn any attribute (property on the class) into a child element by using this pattern:

ElementName.AttributeName (TypeName.PropertyName).

So in this case we assign the enclosed linear gradient brush to the *Fill* property of the *Ellipse*.

1.8: Handling Events of the Controls

Problem Statement:

Handle events of controls.

Solution:

Step 1: Type the following code:

```
<Button Content="Click Me" Click="OnButtonClick">
  <Button.Background>
    <LinearGradientBrush>
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Green" Offset="1" />
    </LinearGradientBrush>
  </Button.Background>
</Button>
```

Example 8: Handle Events of Controls (A)

Note: The `<Button>` element declares that an instance of the `Button` class is created. That `Button` object has its own `Content` property set to the string "Click Me". `Button`'s `Click` event is handled by a method `OnButtonClick` in the code-behind file.

```
<Button Content="Click Me" Click="OnButtonClick">
  <Button.Background>
    <LinearGradientBrush>
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="Yellow" Offset="0" />
        <GradientStop Color="Green" Offset="1" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Button.Background>
</Button>
```

Example 9: Handle Events of Controls (B)



1.9: Using Markup Extensions

Note: The XAML parser also knows how to work with a special construct known as the "markup extension." Markup extensions allow you to compactly configure objects and reference other objects defined elsewhere in the application.

Step 1: Write the following code to test markup extensions:

```
<Grid>
  <Grid.Resources>
    <LinearGradientBrush x:Key="FunkyBrush">
      <GradientStop Color="Yellow" Offset="0" />
      <GradientStop Color="Green" Offset="1" />
    </LinearGradientBrush>
  </Grid.Resources>
  <Button Background="{StaticResource FunkyBrush}">Click
Me</Button>
</Grid>
```

Example 10: Markup Extensions

Note: Notice that the Button's *Background* property is set, via an XML attribute, to *{StaticResource FunkyBrush}*. A *markup extension* is used when an attribute is set to a value which begins with { and ends with }.



1.10: Working With Shapes

Shapes are the core elements of WPF. With shapes, you can draw 2D graphics using shapes such as rectangles, lines, ellipses, paths, polygons, and polylines. These are all represented by classes derived from the abstract base class *Shape*. Shapes are defined in the namespace *System.Windows.Shapes*.

Problem Statement:

Create various shapes in XAML.

Solution:

The following XAML example draws a yellow face with legs, consisting from an ellipse for the face, two ellipses for the eyes, a path for the mouth, and four lines for the legs:

Step 1: Type the following code:

```
<Canvas>
  <Ellipse Canvas.Left="50" Canvas.Top="50" Width="100" Height="100"
    Stroke="Blue" StrokeThickness="4" Fill="Yellow" />
  <Ellipse Canvas.Left="60" Canvas.Top="65" Width="25" Height="25"
    Stroke="Blue" StrokeThickness="3" Fill="White" />
  <Ellipse Canvas.Left="70" Canvas.Top="75" Width="5" Height="5" Fill="Black" />
  <Path Stroke="Blue" StrokeThickness="4" Data="M 62,125 Q 95,122 102,108" />

  <Line X1="124" X2="132" Y1="144" Y2="166" Stroke="Blue" StrokeThickness="4" />
  <Line X1="114" X2="133" Y1="169" Y2="166" Stroke="Blue" StrokeThickness="4" />

  <Line X1="92" X2="82" Y1="146" Y2="168" Stroke="Blue" StrokeThickness="4" />
  <Line X1="68" X2="83" Y1="160" Y2="168" Stroke="Blue" StrokeThickness="4" />
</Canvas>
```

Example 11: Working With Shapes



1.12: Working with Controls

Controls are categorized into these groups:

Control	Description	Examples
Simple Controls	No Content property.	TextBox, PasswordBox, Slider, ScrollBar, etc.
Content Controls	Has a Content property.	Button, Window, Frame, CheckBox, RadioButton, etc.
Headered Content Controls	Content controls with a header.	GroupBox, TabItem, etc.
Items Controls	Contains a list of items that can be accessed with the Items property.	Menu, ListBox, ComboBox, etc.
Headered Items Controls	MenuItem, Toolbar, etc.	

Problem Statement:

Try to use the various controls using XAML as well as via Controls Bar.

1.13: Working with Layouts in XAML

- **StackPanel**

```
<StackPanel>
  <TextBlock>My UI</TextBlock>
  <ListBox>
    <ListBoxItem>Item 1</ListBoxItem>
    <ListBoxItem>Item 2</ListBoxItem>
  </ListBox>
  <RichTextBox/>
</StackPanel>
```

Example 12: StackPanel

- **WrapPanel**

```
<WrapPanel>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
  <Button Width="100">Button</Button>
</WrapPanel>
```

Example 13: WrapPanel

- **DockPanel**

```
<DockPanel>
  <TextBlock DockPanel.Dock="Top">My UI</TextBlock>
  <ListBox DockPanel.Dock="Right">
    <ListBoxItem>Item 1</ListBoxItem>
    <ListBoxItem>Item 2</ListBoxItem>
  </ListBox>
  <RichTextBox/>
</DockPanel>
```

Example 14: DockPanel

- **Canvas**

Canvas is a panel that allows explicitly positioning of controls. Canvas defines the attached properties Left, Right, Top, and Bottom that can be used by the children for positioning within the panel.

```
<Canvas Background="LightBlue">
  <Label Canvas.Top="30" Canvas.Left="20">Enter here:</Label>
  <TextBox Canvas.Top="30" Canvas.Left="130" Width="100"></TextBox>
  <Button Canvas.Top="70" Canvas.Left="130">Click Me!</Button>
</Canvas>
```

Example 15: Canvas



- **Grid**

Using the Grid you can arrange your controls with rows and columns. For every column, you can specify a `ColumnDefinition`, and for every row a `RowDefinition`.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="3*" />
  </Grid.RowDefinitions>

  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>

  <TextBlock Grid.ColumnSpan="2">My UI</TextBlock>
  <ListBox Grid.Row="1" Grid.Column="0">
    <ListBoxItem>Item 1</ListBoxItem>
    <ListBoxItem>Item 2</ListBoxItem>
  </ListBox>
  <RichTextBox Grid.Row="1" Grid.Column="1" />
</Grid>
```

Example 16: Grid



- **Problem Statement (To Do):**

Create a Login Screen (one sample given below) by using the appropriate Layout and various WPF Controls. Also try to use the various features explored so far.

A sample login screen UI. It has a dark red header bar with the text "Log In" in white. Below the header, the text "If you are already a registered user, **Log in** to Centra 7." is displayed. There are two input fields: "Login:" and "Password:". Below the "Password:" field is a checkbox labeled "Remember me". At the bottom is a "Log In" button.



1.14: Working With Styles

Problem Statement:

Creating styles for specific Controls and applying them.

Solution:

The following XAML shows implementation of styles:

```
<StackPanel>
  <StackPanel.Resources>
    <Style TargetType="{x:Type Label}">
      <Setter Property="FontFamily" Value="Trebuchet" />
      <Setter Property="FontSize" Value="12" />
      <Setter Property="FontWeight" Value="Bold" />
    </Style>
  </StackPanel.Resources>

  <Label>Here is some text.</Label>
  <Label>More text.</Label>
  <Label>The last bit of text.</Label>
</StackPanel>
```

Example 17: Working With Styles

Note: The basic and most common part of a Style is its Setters. Simply declare the property name and its value and it will be applied.



Problem Statement:

Apply styles using markup extensions.

Solution:

```
<Window.Resources>
  <Style TargetType="{x:Type Button}">
    <Setter Property="Button.Background" Value="LemonChiffon" />
    <Setter Property="Button.FontSize" Value="18" />
  </Style>
  <Style x:Key="ButtonStyle">
    <Setter Property="Button.Background" Value="AliceBlue" />
    <Setter Property="Button.FontSize" Value="18" />
  </Style>
</Window.Resources>
<StackPanel>
  <Button Name="button2" Width="150">Click Me!</Button>

  <Button Name="button3" Width="150" Style="{StaticResource ButtonStyle}">
    Click Me, Too!
  </Button>
</StackPanel>
```

Example 18: Apply Styles Using Markup Extensions



Problem Statement:

Inherit styles one from another by setting the BasedOn attribute of the style element.

Solution:

```
<StackPanel>
  <StackPanel.Resources>
    <Style x:Key="baseStyle" TargetType="{x:Type Control}">
      <Setter Property="FontFamily" Value="Trebuchet" />
      <Setter Property="FontSize" Value="12" />
      <Setter Property="FontWeight" Value="Bold" />
    </Style>
    <Style BasedOn="{StaticResource baseStyle}" TargetType="{x:Type Label}">
    </Style>
  </StackPanel.Resources>

  <Label>Here is some text.</Label>
  <Label>More text.</Label>
  <Label>The last bit of text.</Label>
</StackPanel>
```

Example 19: Inherit Styles Using BasedOn

Note: Notice that we now have two styles; one that applies to all elements inherited from Control, and a second one based on that Style that extends it – the second Style applies only to Labels.



1.15: Working With Resources

Problem Statement:

Work with Dynamic Resources: With the `StaticResource` markup extension, resources are searched at load time. If the resource changes while the program is running, you should use the `DynamicResource` markup extension instead.

```
<Window.Resources>
  <Style x:Key="ButtonStyle">
    <Setter Property="Button.Background" Value="AliceBlue" />
    <Setter Property="Button.FontSize" Value="18" />
  </Style>
</Window.Resources>

<StackPanel Name="MyContainer">
  <StackPanel.Resources>
    <LinearGradientBrush x:Key="MyGradientBrush" StartPoint="0.5,0" EndPoint="0.5,1">
      <GradientStop Offset="0.0" Color="LightCyan" />
      <GradientStop Offset="0.14" Color="Cyan" />
      <GradientStop Offset="0.7" Color="DarkCyan" />
    </LinearGradientBrush>
  </StackPanel.Resources>

  <Button Name="button1" Width="150">Click Me!</Button>
  <Button Name="button2" Width="150" Style="{StaticResource ButtonStyle}">
    Click Me, Too!
  </Button>
  <Button Name="button3" Width="200" Height="50" Click="OnChangeResource">
    Change Resource
  </Button>
  <Button Name="button4" Width="200" Height="50"
    Background="{DynamicResource MyGradientBrush}">
    Dynamic Resource
  </Button>
</StackPanel>
```

Example 20: Working With Dynamic Resources



Write the following code on the event handler *OnChangeResource*:

```
private void OnChangeResource(object sender, RoutedEventArgs e)
{
    MyContainer.Resources.Clear();
    LinearGradientBrush brush = new LinearGradientBrush();
    brush.StartPoint = new Point(0.5, 0);
    brush.EndPoint = new Point(0.5, 1);
    GradientStopCollection stops = new GradientStopCollection();
    stops.Add(new GradientStop(Colors.White, 0.0));
    stops.Add(new GradientStop(Colors.Yellow, 0.14));
    stops.Add(new GradientStop(Colors.YellowGreen, 0.7));
    brush.GradientStops = stops;
    MyContainer.Resources.Add("MyGradientBrush", brush);
}
```

Example 21: OnChangeResource Code

1.16: Working With Triggers

A Trigger is a stateful aspect of the Style. Define a Trigger so that when the user moves with the mouse over a button, the button should change its look. Usually, you have had to do this with the C# code; with WPF you can also do this with XAML.

Work With Triggers

```
<Button MinWidth="75" Margin="10">
  <Button.Style>
    <Style TargetType="{x:Type Button}">
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter Property="Foreground" Value="Blue"/>
        </Trigger>
      </Style.Triggers>
    </Style>
  </Button.Style>
  OK
</Button>
```

Example 22: Triggers (A)

Another Example of Triggers



```
<StackPanel>
  <StackPanel.Resources>
    <Style x:Key="baseStyle" TargetType="{x:Type Control}">
      <Setter Property="FontFamily" Value="Trebuchet" />
      <Setter Property="FontSize" Value="12" />
      <Setter Property="FontWeight" Value="Bold" />
    </Style>
    <Style BasedOn="{StaticResource baseStyle}" TargetType="{x:Type Label}">
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter Property="Foreground" Value="Red" />
        </Trigger>
      </Style.Triggers>
    </Style>
  </StackPanel.Resources>

  <Label>Here is some text.</Label>
  <Label>More text.</Label>
  <Label>The last bit of text.</Label>
</StackPanel>
```

Example 23: Triggers (B)

Lab Summary:

In this lab session you became familiar with XAML and the various new features WPF offers.



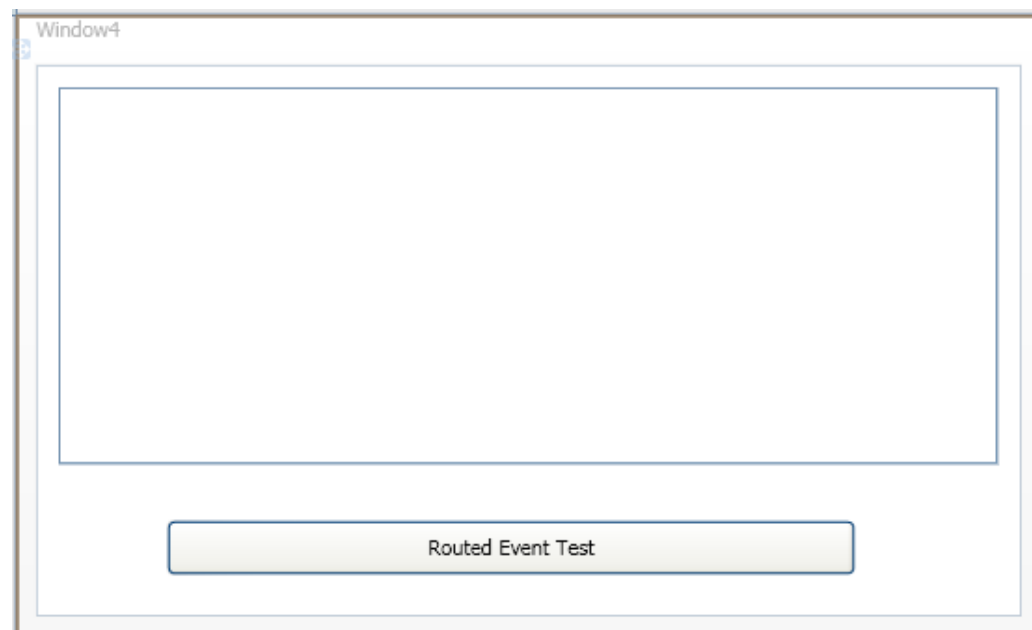
Goals	<ul style="list-style-type: none">Understanding Routed Events
Time	90 minutes

Step 1. Add a new WPF Window Project => Add Window, Name it RoutedEventDemo.xaml

Step 2. Put the following xaml

Note: Preview and Click events are being handled at all levels

```
<Window x:Class="IGatePatniApp.RoutedEventDemo"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window4" Height="333" Width="550" Button.Click="Window_Click"
  PreviewMouseDown="Window_PreviewMouseDown"
  PreviewMouseUp="Window_PreviewMouseUp">
  <Grid PreviewMouseDown="Grid_PreviewMouseDown" Button.Click="Grid_Click"
    PreviewMouseUp="Grid_PreviewMouseUp">
    <ListBox Margin="12,12,12,81" Name="listBox1" />
    <Button Margin="70,0,88,21"
      PreviewMouseDown="btneventtest_PreviewMouseDown"
      PreviewMouseUp="btneventtest_PreviewMouseUp"
      Click="btneventtest_Click"
      Name="btneventtest" Height="30" VerticalAlignment="Bottom">Routed Event
    Test</Button>
  </Grid>
</Window>
```

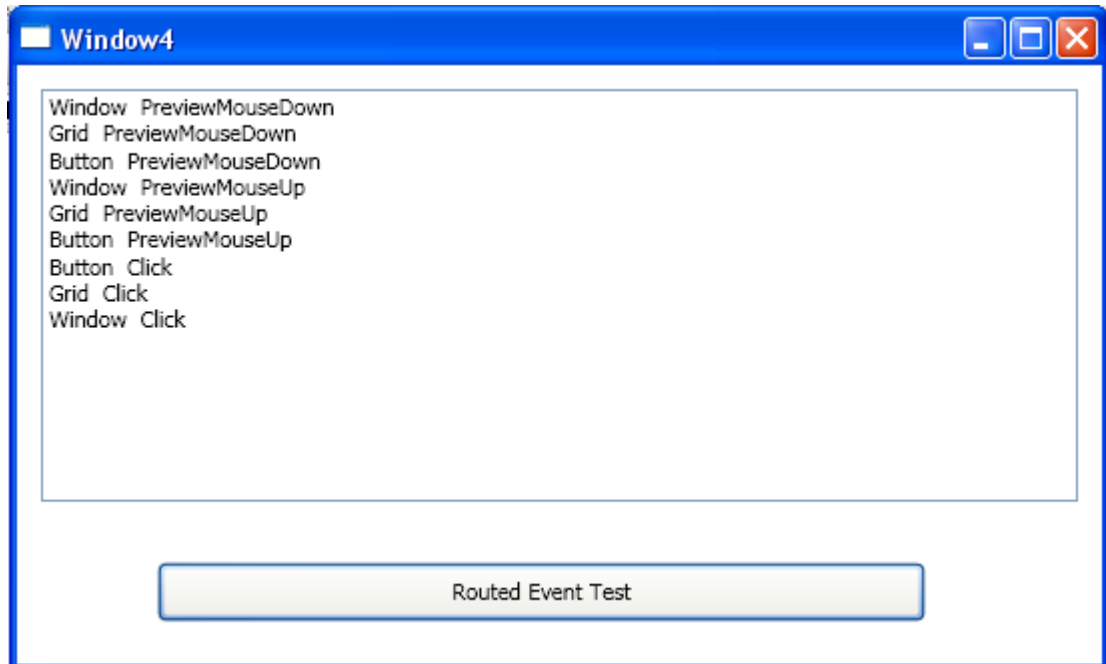


Step 3. The Event Handlers defined in the code behind



```
private void Window_PreviewMouseDown(object sender,
                                     MouseButtonEventArgs e)
{
    listBox1.Items.Clear();
    listBox1.Items.Add("Window " + e.RoutedEvent.Name);
}
private void Window_PreviewMouseUp(object sender,
                                    MouseButtonEventArgs e)
{
    listBox1.Items.Add("Window " + e.RoutedEvent.Name);
}
private void Grid_PreviewMouseDown(object sender,
                                    MouseButtonEventArgs e)
{
    listBox1.Items.Add("Grid " + e.RoutedEvent.Name);
}
private void Grid_PreviewMouseUp(object sender,
                                  MouseButtonEventArgs e)
{
    listBox1.Items.Add("Grid " + e.RoutedEvent.Name);
}
private void Grid_Click(object sender, RoutedEventArgs e)
{
    listBox1.Items.Add("Grid " + e.RoutedEvent.Name);
}
private void btneventtest_PreviewMouseDown(object sender,
                                             MouseButtonEventArgs e)
{
    listBox1.Items.Add("Button " + e.RoutedEvent.Name);
}
private void btneventtest_PreviewMouseUp(object sender,
                                           MouseButtonEventArgs e)
{
    listBox1.Items.Add("Button " + e.RoutedEvent.Name);
}
private void btneventtest_Click(object sender, RoutedEventArgs e)
{
    listBox1.Items.Add("Button " + e.RoutedEvent.Name);
}
private void Window_Click(object sender, RoutedEventArgs e)
{
    listBox1.Items.Add("Window " + e.RoutedEvent.Name);
}
```

Step 4. Make this page as startup and run. click on the button and see the list of events in the listbox.



Note:- After clicking button do not click anywhere because listbox will be reloaded with the new set of events



Goals	<ul style="list-style-type: none">• Build a WPF application which uses Data Binding• Using Database Connectivity
Time	120 minutes

The Database Table used for this Lab:

```
create table cust
(
    custid varchar(10) primary key,
    custname varchar(50),
    credlimit numeric(10,2),
    IsPremium bit
)

insert into cust values('an','anand',10000,1)
insert into cust values('mil','milind',15000,1)
insert into cust values('sac','sachin',25000,0)
```

- Step 1.** Create a new WPF application.
- File => New Project, Select WPF Application Template, Name it IGatePatni.WPF.Application
- Step 2.** Add Application Configuration File
- Project => Add New Item, Select Application Configuration File template. Let the name be App.config
- Step 3.** Add following Connection String related xml entry in App.config within <configuration></configuration>

```
<connectionStrings>
<add name="labdemoconnectstring"
connectionString="server=atrgsql\sql2005;database=labdemos;user
id=sqluser;password=sqluser"/>
</connectionStrings>
```

- Step 4.** Add a class called CustomerService
- Project => Add class.

Following is the code for the class:

```
using System.Data;
using System.Data.SqlClient;

public class CustomerService
{
    public void GetData(string custid, Customer ce)
    {
        string connectstring = ConfigurationManager.ConnectionStrings
            ["labdemoconnectstring"].ConnectionString;
        SqlConnection con = new SqlConnection(connectstring);
```

```

SqlCommand cmd = new SqlCommand
    ("select * from cust where custid = " + custid + "", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
if (dr.Read())
{
    ce.CustID = dr["custid"].ToString();
    ce.CustName = dr["custname"].ToString();
    ce.CreditLimit =
        double.Parse(dr["credlimit"].ToString());
    ce.Premium = bool.Parse(dr["ispremium"].ToString());
}
dr.Close(); con.Close();
}

public void SaveChanges(Customer c)
{
    string connectsring = ConfigurationManager.ConnectionStrings
        ["labdemoconnectstring"].ConnectionString;
    SqlConnection con = new SqlConnection(connectsring);

    string sql = "update cust set custname = " + c.CustName + ",credlimit = " +
        c.CreditLimit.ToString() + ",ispremium = "
            + (c.Premium == true ? 1 : 0).ToString()
            + " where custid = " + c.CustID + "";
    SqlCommand cmd = new SqlCommand(sql, con);
    con.Open();
    cmd.ExecuteNonQuery();
}
}

```

Step 5. Add one more class definition, named Customer, and write the following code:

```

using System.Windows;

public class Customer : DependencyObject
{
    protected static DependencyProperty CustIDProperty =
        DependencyProperty.Register
            ("CustID", typeof(string), typeof(Customer));

    protected static DependencyProperty CustNameProperty =
        DependencyProperty.Register
            ("CustName", typeof(string), typeof(Customer));

    protected static DependencyProperty CreditLimitProperty =
        DependencyProperty.Register
            ("CreditLimit", typeof(double), typeof(Customer));
}

```

```

        protected static DependencyProperty PremiumProperty =
        DependencyProperty.Register
            ("Premium", typeof(bool), typeof(Customer));

        public string CustID
        {
            get { return GetValue(Customer.CustIDProperty).ToString(); }
            set
            {
                SetValue(Customer.CustIDProperty, value);
            }
        }
        public string CustName
        {
            get { return GetValue(Customer.CustNameProperty).ToString(); }
            set
            {
                SetValue(Customer.CustNameProperty, value);
            }
        }
        public double CreditLimit
        {
            get { return (double)GetValue(Customer.CreditLimitProperty); }
            set
            {
                SetValue(Customer.CreditLimitProperty, value);
            }
        }
        public bool Premium
        {
            get { return (bool)GetValue(Customer.PremiumProperty); }
            set
            {
                SetValue(Customer.PremiumProperty, value);
            }
        }

        public void GetData()
        {
            CustomerService cs = new CustomerService();
            cs.GetData(this.CustID, this);
            this.IsSaveEnabled = true;
        }
        public void SaveChanges()
        {
            CustomerService cs = new CustomerService();
            cs.SaveChanges(this);
            this.IsSaveEnabled = false;
        }
    }

```



#endregion

Step 6. Open window1.xaml and change the xaml code to following:
Note the shaded xaml instruction lines

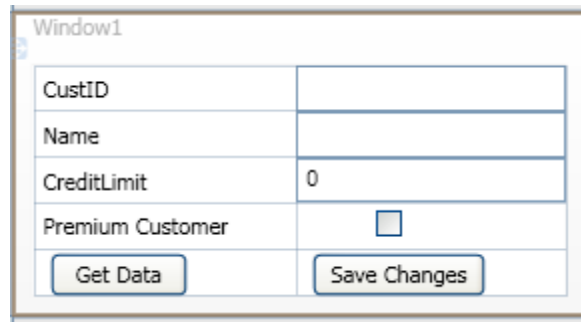
```
<Window x:Class=" IGatePatni.WPF.Application.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="152" Width="288" SizeToContent="WidthAndHeight"
xmlns:cs="clr-namespace: IGatePatni.WPF.Application" >
<Window.Resources>
  <cs:Customer x:Key="custentity" />
</Window.Resources>
<Grid DataContext="{StaticResource custentity}" >
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="138*" />
    <ColumnDefinition Width="141*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>

  <Label Name="label1">CustID</Label>
  <Label Grid.Row="1" Name="label2">Name</Label>
  <Label Grid.Row="2" Name="label3">CreditLimit</Label>
  <Label Grid.Row="3" Name="label4">Premium Customer</Label>

  <TextBox Grid.Column="1" Name="tcustid" Text="{Binding Path=CustID ,
Mode=TwoWay}" />
  <TextBox Grid.Column="1" Grid.Row="1" Name="tcustname" Text="{Binding
Path=CustName , Mode=TwoWay }" />
  <TextBox Grid.Column="1" Grid.Row="2" Name="tcrelimit" Text="{Binding
Path=CreditLimit , Mode=TwoWay}" />
  <CheckBox Name="chkpremium" Margin="40,4" Grid.Column="1" Grid.Row="3"
IsChecked="{Binding Path=Premium , Mode=TwoWay}"/>

  <Button Grid.Row="4" Margin="8,2,54,-1" Name="btnget"
Click="btnget_Click">Get Data</Button>
  <Button Grid.Column="1" Grid.Row="4" Margin="8,2,54,-1"
Name="btnsave" Click="btnsave_Click">Save Changes</Button>
</Grid>
</Window>
```

The XAML window should look as below:



Step 7. Write the following code in the code behind:

```
private void btnget_Click(object sender, RoutedEventArgs e)
{
    Customer c = (Customer)this.FindResource("custentity");
    c.GetData();
}

private void btnsave_Click(object sender, RoutedEventArgs e)
{
    Customer c = (Customer)this.FindResource("custentity");
    c.SaveChanges();
    MessageBox.Show("Changes Saved");
}
```

Assignment To Do:

Comment the existing definition of "CreditLimit" property in the above lab and define that property as DependencyProperty.



Goals	<ul style="list-style-type: none">• Binding WPF GUI with a collection and navigate collection Items.
Time	90 minutes

Step 5. Add a new WPF Window Project => Add Window, Name it CollectionBindDemo.xaml

Step 6. Add a class Definition file Product.cs

```
public class Product
{
    public string ProductName { get; set; }
    public double ProductStock { get; set; }
}
```

Step 7. Add one more class Definition file ProductList.cs

```
public class ProductList : ObservableCollection<Product>
{
    public ProductList()
    {
        Product p = new Product();
        p.ProductName = "Black Berry"; p.ProductStock = 100;
        this.Add(p);

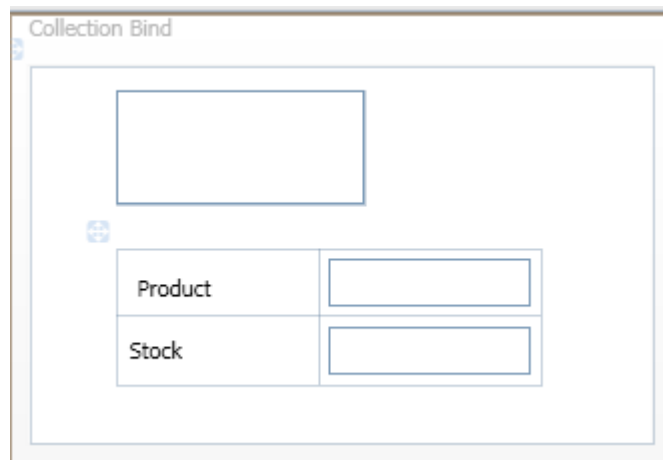
        p = new Product();
        p.ProductName = "Samsung Galaxy"; p.ProductStock = 200;
        this.Add(p);

        p = new Product();
        p.ProductName = "Samsung Wave"; p.ProductStock = 300;
        this.Add(p);
    }
}
```

Step 8. The XAML for the CollectionBindDemo.xaml is given below:



```
<Window x:Class="WPFDay2.CollectionBindDemo"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Collection Binding" Height="226" Width="333" Loaded="Window_Loaded" >
  <Canvas x:Name="can">
    <ListBox IsSynchronizedWithCurrentItem="True" Canvas.Left="43"
      Canvas.Top="12" ItemsSource="{Binding}" DisplayMemberPath="ProductName"
      Height="57" Width="124" />
    <Grid Height="68" Width="212" Canvas.Left="43" Canvas.Top="91">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="114*" />
        <ColumnDefinition Width="124*" />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
      </Grid.RowDefinitions>
      <TextBox Margin="5" Text="{Binding Path= ./ProductStock, Mode=TwoWay}"
        Grid.Column="1"
        Grid.Row="1" />
      <TextBox Margin="5" Text="{Binding Path= ./ProductName, Mode=TwoWay}"
        Grid.Column="1" />
      <Label Margin="6,8,10,5" Name="label1">Product</Label>
      <Label Grid.Row="1" Margin="2,5,10,6" Name="label2">Stock</Label>
    </Grid>
  </Canvas>
</Window>
```



Step 9. The code for window loaded event:

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    ProductList pl = new ProductList();
    can.DataContext = pl;
}
```



}
}

Step 10.

Change the startup window to CollectionBindDemo in App.xaml and run

- i. Note: `IsSynchronizedWithCurrentItem="True"` property setting for listbox. You change the listbox selection to get the details of a product.
- ii. You can change product stock of "Samsung Galaxy" to 500 navigate to "Samsung Wave" and navigate back to "Samsung Galaxy", The changes will be applied to product details in collection.



Lab 2. Develop Rich User Interface With XAML

Goals	<ul style="list-style-type: none">• Build a rich user interface that help understand WPF features in detail.
Time	90 minutes

Exercise 2.1: Design GUI

In this exercise, you will develop a Magical Question Box Application. The task is to design an application to accept a question from the user and provide the answer from the list of answers stored as a collection.

Task 1: Type the following code:

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Grid.Background>
    <LinearGradientBrush>
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0.00" Color="Red" />
        <GradientStop Offset="0.50" Color="Indigo" />
        <GradientStop Offset="1.00" Color="Violet" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Grid.Background>
  <TextBox VerticalAlignment="Stretch" HorizontalAlignment="Stretch" Margin="10,10,13,10" Name="txtQuestion"
    TextWrapping="Wrap" FontFamily="Verdana" FontSize="24"
    Grid.Row="0" >
    [Place question here.]
  </TextBox>
  <Button VerticalAlignment="Top" HorizontalAlignment="Left" Margin="10,0,0,20" Width="127" Height="23"
    Name="cmdAnswer"
    Click="cmdAnswer_Click"
    Grid.Row="1">
    Ask the Eight Ball
  </Button>
  <TextBox VerticalAlignment="Stretch" HorizontalAlignment="Stretch" Margin="10,10,13,10" Name="txtAnswer"
    TextWrapping="Wrap" IsReadOnly="True" FontFamily="Verdana" FontSize="24" Foreground="Green"
    Grid.Row="2">
    [Answer will appear here.]
  </TextBox>
</Grid>
</Window>
```

Example 24: Code to Design GUI



Task 2: Add a class in the project. This class provides answers to questions asked by the user. Add the following code:

```
class AnswerGenerator
{
    string[] answers = new string[]{
        "Ask Again later", "Can Not Predict Now", "Without a Doubt", "Is Decidedly So", "Concentrate
        and Ask Again", "My Sources Say No", "Yes, Definitely", "Don't Count On It", "Signs Point to
        Yes", "Better Not Tell You Now", "Outlook Not So Good", "Most Likely", "Very Doubtful", "As I See
        It, Yes", "My Reply is No", "It Is Certain", "Yes", "You May Rely On It", "Outlook Good", "Reply
        Hazy Try Again"
    };

    public string GetRandomAnswer(string question)
    {
        Random rnd = new Random();

        return answers[rnd.Next(0, answers.Length)];
    }
}
```

Example 25: Add Class in Project

Task 3: Write the event Handler for the Button. This event handler will be generating the random answer and display in the grid. Write the following code:

```
private void cmdAnswer_Click(object sender, RoutedEventArgs e)
{
    // Dramatic delay...
    this.Cursor = Cursors.Wait;
    System.Threading.Thread.Sleep(TimeSpan.FromSeconds(1));

    AnswerGenerator generator = new AnswerGenerator();
    txtAnswer.Text = generator.GetRandomAnswer(txtQuestion.Text);
    this.Cursor = null;
}
```

**Example 26: Add Click Event for Button**

Task 4: Execute the Application and check the output.

Exercise 2.2: Design a Window That Uses Event Trigger Features

Task 1: Define the styles and Style Triggers. Add the following code in a new window:

```
<Window.Resources>
  <Style x:Key="BigFontButton">
    <Style.Setters>
      <Setter Property="Control.FontFamily" Value="Times New Roman" />
      <Setter Property="Control.FontSize" Value="18" />
      <Setter Property="Control.FontWeight" Value="Bold" />
    </Style.Setters>

    <Style.Triggers>
      <EventTrigger RoutedEvent="Mouse.MouseEnter">
        <EventTrigger.Actions>
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation
                Duration="0:0:0.2"
                Storyboard.TargetProperty="FontSize"
                To="22" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger.Actions>
      </EventTrigger>
      <EventTrigger RoutedEvent="Mouse.MouseLeave">
        <EventTrigger.Actions>
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation
                Duration="0:0:1"
                Storyboard.TargetProperty="FontSize" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger.Actions>
      </EventTrigger>
    </Style.Triggers>
  </Style>
</Window.Resources>
```

Example 27: Styles and Style Triggers



Task 2: Add *three* buttons on the window and apply the styles defined above. Write the following code:

```
<StackPanel Margin="5">
    <Button Padding="5" Margin="5"
        Style="{StaticResource BigFontButton}"
        >A Customized Button</Button>
    <TextBlock Margin="5">Normal Content.</TextBlock>
    <Button Padding="5" Margin="5"
        >A Normal Button</Button>
    <TextBlock Margin="5">More normal Content.</TextBlock>
    <Button Padding="5" Margin="5"
        Style="{StaticResource BigFontButton}"
        >Another Customized Button</Button>
</StackPanel>
```

Example 28: Three Buttons to Apply Styles

Task 3: Build and execute the application.



Appendices

Appendix A: Table of Examples

Example 1: Do As Instructed	5
Example 2: Generate Button	6
Example 3: Use Properties as Attributes	6
Example 4: Properties as Elements	7
Example 5: Style To Set Complex Property Values	7
Example 6: Attached Properties	8
Example 7: Property Element Syntax	8
Example 8: Handle Events of Controls (A)	9
Example 9: Handle Events of Controls (B)	9
Example 10: Markup Extensions	10
Example 11: Working With Shapes	11
Example 12: StackPanel	12
Example 13: WrapPanel	13
Example 14: DockPanel	13
Example 15: Canvas	13
Example 16: Grid	14
Example 17: Working With Styles	16
Example 18: Apply Styles Using Markup Extensions	17
Example 19: Inherit Styles Using BasedOn	18
Example 20: Working With Dynamic Resources	19
Example 21: OnChangeResource Code	20
Example 22: Triggers (A)	20
Example 23: Triggers (B)	21
Example 24: Code to Design GUI	33
Example 25: Add Class in Project	34
Example 26: Add Click Event for Button	35
Example 27: Styles and Style Triggers	35
Example 28: Three Buttons to Apply Styles	36