* WPF - Windows Presentation Foundation
* Used in designing UI/Development Based Application
* Client Based Application
* Used in Animation, styling, 2D-3D
* XAML – Extensible Application Markup Language
* **Create Project -> Search WPF -> Select “WPF Application with C#, Windows, Desktop”.**
* MainWindow.xaml -> MainWindow.xaml.cs (It has a partial class which combine it with xaml file), (Window is the root tag)
* App.xaml(Root File) +App.xaml.cs

**Note:** In App.xaml.cs file, (StartupUri =” MainWindow.xaml”) means **MainWindow.xaml** is the starting point..

**In MainWindow.xaml->**

<Window x:Class="KFCApp.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KFCApp"

mc:Ignorable="d"

Title="KFC" Height="450" Width="800" Icon="kfclogo.jfif" ResizeMode="CanResize"

ShowInTaskbar="True">

<Grid>

<Label Content="Hello KFCians!!"/>

<TextBlock Margin="10" Foreground="Red">This is a Text Block1..</TextBlock>

<TextBlock Margin="10" TextTrimming="CharacterEllipsis" Foreground="Green">This is a Text Block2..</TextBlock>

<TextBlock Margin="10" TextWrapping="Wrap" Foreground="Blue">This is a Text Block3..</TextBlock> </Grid>

</Window>

**Notes:**

* For Icon/Images, Left click on Icon -> Properties -> Build Action = Resource
* **ShowInTaskbar** is true by default.
* **Grid** = Tabular format
* **Label** can hold/display image & display in single line
* **TextBlock** is used only for string, can display in single or multi-line
* We can use **Grid** or **StackPanel** at a time and in grid we have to use **</LineBreak>** for next line and for StackPanel we don’t need anything.
* We can take StackPanel inside StackPanel

**In MainWindow.xaml.cs->**

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace KFCApp

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

}

}

**In App.xaml->**

<Application x:Class="KFCApp.App"

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

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

xmlns:local="clr-namespace:KFCApp"

StartupUri="MainWindow.xaml">

<Application.Resources>

</Application.Resources>

</Application>

**Note:** MainWindow.xaml is the starting file

**Note:** We can call from code too by creating a method ex: WindowStart in App.xaml.cs (Startup="WindowStart")

**In App.xaml.cs->**

using System.Configuration;

using System.Data;

using System.Windows;

namespace KFCApp

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

private void WindowStart(Object sender, StartupEventArgs e)

{

MainWindow winobj = new MainWindow();

winobj.Title = "-KFC-";

winobj.Show();

}

}

}

**Note:** Created a method i.e WindowStart to make it a starting point.

**Note:** MainWindow is a partial class which we called from MainWindow.xaml.cs

**TextBlock Control using StackPanel:**

**In MainWindow.xaml->**

<Window x:Class="KFCApp.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KFCApp"

mc:Ignorable="d"

Title="KFC" Height="450" Width="800" Icon="kfclogo.jfif" ResizeMode="CanResize"

ShowInTaskbar="True">

<StackPanel>

<Label Content="Hello KFCians!!"/>

<TextBlock Margin="10" Foreground="Red">This is a Text Block1..</TextBlock>

<TextBlock Margin="10" TextTrimming="CharacterEllipsis" Foreground="Green">This is a Text Block2..</TextBlock>

<TextBlock Margin="10" TextWrapping="Wrap" Foreground="Blue">This is a Text Block3..</TextBlock>

</StackPanel>

</Window>

**CheckBox Control using StackPanel with IsThreeState Property:**

**Note:** CheckBox has 2 state - True & False.

**Note:** IsThreeState property is used for checking all checkbox and have 3 state - true, false, null.

**In MainWindow.xaml->**

<Window x:Class="KFCApp.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KFCApp"

mc:Ignorable="d"

Title="KFC" Height="450" Width="800" Icon="kfclogo.jfif" ResizeMode="CanResize"

ShowInTaskbar="True">

<StackPanel>

<Label Content="Hello KFCians!!"/>

<Label>CheckBox 1:</Label>

<CheckBox>Wpf</CheckBox>

<CheckBox> UWP</CheckBox>

<CheckBox IsChecked="True">ASP.NET</CheckBox>

<StackPanel Margin="10">

<Label>CheckBox 2</Label>

<CheckBox>

<TextBlock>Hello</TextBlock>

</CheckBox>

</StackPanel>

<StackPanel Margin="10">

<Label>CheckBox 3:</Label>

<CheckBox IsThreeState="True" Name="AllCheck" Checked="AllCheck\_Checked" Unchecked="AllCheck\_Unchecked">

<TextBlock>All Features enabled</TextBlock>

</CheckBox>

<StackPanel Margin="20,5">

<!--<Label>CheckBox 4 under checkbox 3:</Label>-->

<CheckBox IsThreeState="True" Name="First" Checked="first\_Checked" Unchecked="first\_Checked">First CheckBox</CheckBox>

<CheckBox IsThreeState="True" Name="Second" Checked="first\_Checked" Unchecked="first\_Checked">Second CheckBox</CheckBox>

<CheckBox IsThreeState="True" Name="Third" Checked="first\_Checked" Unchecked="first\_Checked">Third CheckBox</CheckBox>

</StackPanel>

</StackPanel>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace KFCApp

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void first\_Checked(object sender, RoutedEventArgs e)

{

AllCheck.IsChecked = null;

if(First.IsChecked == true && Second.IsChecked == true && Third.IsChecked == true)

{

AllCheck.IsChecked = true;

}

if (First.IsChecked == false && Second.IsChecked == false && Third.IsChecked == false)

{

AllCheck.IsChecked = false;

}

}

private void first\_Unchecked(object sender, RoutedEventArgs e)

{

}

private void AllCheck\_Checked(object sender, RoutedEventArgs e)

{

bool allstate = (AllCheck.IsChecked == true);

First.IsChecked = allstate;

Second.IsChecked = allstate;

Third.IsChecked = allstate;

}

private void AllCheck\_Unchecked(object sender, RoutedEventArgs e)

{

}

}

}

A screenshot of a computer

Description automatically generated

**Button Control (with Image Inside it & Click Event):**

**A logo of a person

Description automatically generated A grey rectangle with black text

Description automatically generated**

**With Click Event:**

**In MainWindow.xaml->**

<Window x:Class="KFCApp2.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KFCApp2"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<Button x:Name="Clickbtn" Content ="Click Me!!!" Height="100" Width="160" Click="Clickbtn\_Click">

</Button>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace KFCApp2

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

Clickbtn.Content = "Save Me...";

}

}

}

**With Image Button:**

**Note:** For Icon/Images, Left click on Icon -> Properties -> Build Action = Resource

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<Button x:Name="clickbtn" Height="100" Width="160" Click="Button\_Click">

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\kfclogo.jfif"></Image>

</Button>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace KFCApp2

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

Clickbtn.Content = "Save Me...";

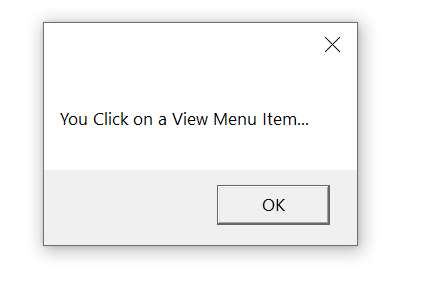
}

}

}

**Menu(Menu Item, Sub Menu Item, Event Handling, Check Box, Image):**

**A screenshot of a computer menu

Description automatically generated **

**Notes:**

* Add a new xaml file in your project i.e right click on project -> New Item -> select Window(WPF) & name it as menu.xaml -> Add.
* Change Startup point to menu.xaml i.e Go to App.xaml -> Edit StartUri to menu.xaml i.e StartupUri="menu.xaml"
* To display menu from main header we have to use IsMainMenu="True"
* If we want to access menu item with app then we have to use underscore with MenuItemName ”\_MenuItem” ex= <MenuItem Header="\_File"/>
* MessageBox is a prompt box with a msg when you click on a menu item.

**Example:**

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

{

MessageBox.Show("You Click on File Menu Item...");

}

**Note:** Command is used for shortcut which creates automatically like (ctrl + n\*) i.e <MenuItem Header="New" Command="New"/>

**Note:** MenuItem.Icon is used for Icon for menu item.

**Example:**

<MenuItem Header="\_FontEdit">

<MenuItem.Icon>

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\kfclogo.jfif"/>

</MenuItem.Icon>

</MenuItem>

**Note:** For CheckBox we have to add IsCheckable="True" IsChecked="True" in MenuItem. Example: <MenuItem Header="\_FileEdit" IsCheckable="True" IsChecked="True"/>

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.menu"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="menu" Height="450" Width="800">

<Grid>

<StackPanel>

<Menu IsMainMenu="True">

<MenuItem Header="\_File" Click="MenuItem\_Click">

<MenuItem Header="New" Command="New"/>

</MenuItem>

<MenuItem Header="\_Edit">

<MenuItem Header="\_FontEdit">

<MenuItem.Icon>

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\kfclogo.jfif"/>

</MenuItem.Icon>

</MenuItem>

<MenuItem Header="\_FileEdit" IsCheckable="True" IsChecked="True"/>

<MenuItem Header="\_SaveEdit"/>

</MenuItem>

<MenuItem Header="\_View" Click="MenuItem\_Click"/>

<MenuItem Header="\_Help"/>

</Menu>

</StackPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for menu.xaml

/// </summary>

public partial class menu : Window

{

public menu()

{

InitializeComponent();

}

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

{

MessageBox.Show("You Click on a File Menu Item...");

}

private void MenuItem\_Click\_1(object sender, RoutedEventArgs e)

{

MessageBox.Show("You Click on a View Menu Item...");

}

}

}

**Context Menu with RichTextBox:**

**A blue rectangle with black text

Description automatically generated**

**Notes:**

* RichTextBox is used for editing in Client Page like a messaging tool.
* We have used “Cut” command in below program which can be use to cut and paste the text from RichTextBox. Select Text -> Right Click on Selected Text -> Cut
* We can give Icon in ContextMenu MenuItem too.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ContextMenu"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="ContextMenu" Height="450" Width="800">

<Grid>

<TextBlock HorizontalAlignment="Center" Margin="0,135,0,10">Type Something here!!!</TextBlock>

<RichTextBox Height="100" Width="200" BorderThickness="4" BorderBrush="Black" Background="Lavender">

<RichTextBox.ContextMenu>

<ContextMenu>

<MenuItem Header="Cut" Command="Cut"/>

<MenuItem Header="Copy" Command="Copy"/>

<MenuItem Header="Paste" Command="Paste">

<MenuItem.Icon>

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\kfclogo.jfif"/>

</MenuItem.Icon>

</MenuItem>

</ContextMenu>

</RichTextBox.ContextMenu>

</RichTextBox>

</Grid>

</Window>

**Expander Control:**

**A screenshot of a computer

Description automatically generated**

**Note:** The Expander control will provide you with the ability to **hide/show a piece of content**. This would usually be a piece of text, but thanks to the flexibility of WPF, it can be used for any type of mixed content like texts, images and even other WPF controls.

**Note:** <Grid.RowDefinitions>

<RowDefinition>

</RowDefinition>

</Grid.RowDefinitions>

Grid.RowDefinitions and RowDefinitions is used for aligning/wrapping the text.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ExpanderControl"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="ExpanderControl" Height="450" Width="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition>

</RowDefinition>

</Grid.RowDefinitions>

<Expander Header="Fruits List" Grid.Row="0">

<StackPanel>

<Label>Mango</Label>

<Label>Apple</Label>

<Label>Banana</Label>

<Label>Orange</Label>

<Expander Header="Vegetable List" Grid.Row="1">

<StackPanel>

<Label>Potato</Label>

<Label>Tomato</Label>

<Label>Ginger</Label>

</StackPanel>

</Expander>

<Expander ExpandDirection="Right">

<TextBlock TextWrapping="Wrap" FontSize="18">

Exapnder in a Expander Example: Here we can have text which can be hidden/shown using the built-in functionality of the Expander control.

</TextBlock>

</Expander>

</StackPanel>

</Expander>

</Grid>

</Window>

**DatePicker Control:**

**A close-up of a computer screen

Description automatically generated**

**Note:** The DatePicker control will be displayed pretty much like a regular TextBox, but with a small button which will bring up a Calendar-view when clicked, allowing your user to select the date.

**Example using xaml:**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DatePickerControl"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DatePickerControl" Height="450" Width="800">

<Grid>

<StackPanel Margin="111,111,111,111">

<DatePicker>

</DatePicker>

<DatePicker DisplayDateStart="2024-02-21" DisplayDateEnd="2024-02-28" FirstDayOfWeek="Monday" IsTodayHighlighted="True">

</DatePicker>

</StackPanel>

</Grid>

</Window>

**Example using Code Behind File:**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DatePickerControl"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DatePickerControl" Height="450" Width="800">

<Grid>

<StackPanel Margin="111,111,111,111" x:Name="sp1">

</StackPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

public partial class DatePickerControl : Window

{

public DatePickerControl()

{

InitializeComponent();

DatePicker dp1 = new DatePicker();

dp1.Name = "datepicker";

dp1.DisplayDateStart = DateTime.Now;

dp1.DisplayDateEnd = new DateTime(2024,03,22);

dp1.IsTodayHighlighted = true;

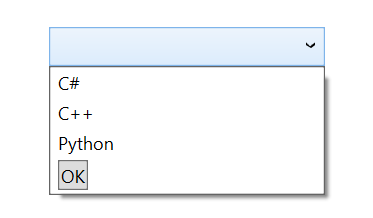
sp1.Children.Add(dp1);

}

}

}

**ComboBox:**

****  **A screenshot of a computer error

Description automatically generated**

**Note:** ComboBox is a type of collection where we can display single item at a time.

**Note:** Drag & Paste ComboBox from tool box -> Click on ComboBox and go to properties -> Click on Items (Collection) -> Select Object from last column -> Search String and Choose String -> Click on Add -> Select String in System from mscorlib & press ok

Like that Add button too - Click on Items (Collection) -> Select Button & Add -> Select Button from System.Windows.Controls -> drop down to last column -> add content = “Ok” in </Button>

**Note:** SelectionChanged = popup tool

**In (MainWindow.xaml->**

<Window

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

xmlns:System="clr-namespace:System;assembly=System.Runtime" x:Class="KfcApp2.ComboBox"

mc:Ignorable="d"

Title="ComboBox" Height="450" Width="800">

<Grid>

<ComboBox x:Name="combo1" SelectionChanged="ComboBox\_SelectionChanged" HorizontalAlignment="Center" Margin="0,168,0,0" VerticalAlignment="Top" Width="184" RenderTransformOrigin="-0.741,0.302" Height="26">

<System:String>C#</System:String>

<System:String>C++</System:String>

<System:String>Python</System:String>

<Button Content="OK"/>

</ComboBox>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for ComboBox.xaml

/// </summary>

public partial class ComboBox : Window

{

public ComboBox()

{

InitializeComponent();

}

private void ComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

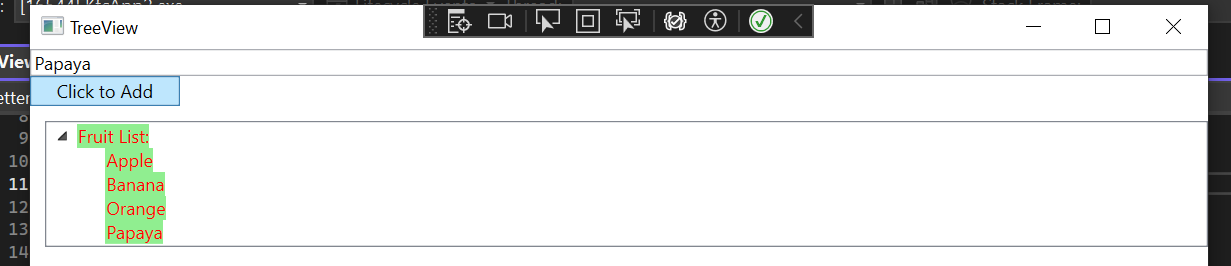
MessageBox.Show("You selected " + combo1.SelectedItem.ToString());

}

}

}

**Tree View, Style using <Window.Resources> & Setter Property:**

****

**Note:** TreeView Property = “TreeViewItem”

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.TreeView"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="TreeView" Height="450" Width="800">

<Window.Resources>

<Style TargetType="{x:Type TreeViewItem}">

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

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

</Style>

</Window.Resources>

<Grid>

<StackPanel>

<TextBox x:Name="Treetxt"/>

<Button Content="Click to Add" Click="Button\_Click" Width="100" HorizontalAlignment="Left"/>

<TreeView Margin="10,10,0,10" Name="TreeView1">

<TreeViewItem Header="Fruit List:" x:Name="FruitListItem">

<TreeViewItem Header="Apple"></TreeViewItem>

<TreeViewItem Header="Banana"></TreeViewItem>

<TreeViewItem Header="Orange"></TreeViewItem>

</TreeViewItem>

</TreeView>

</StackPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for TreeView.xaml

/// </summary>

public partial class TreeView : Window

{

public TreeView()

{

InitializeComponent();

}

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

{

TreeViewItem childitem = new TreeViewItem();

childitem.Header = Treetxt.Text;

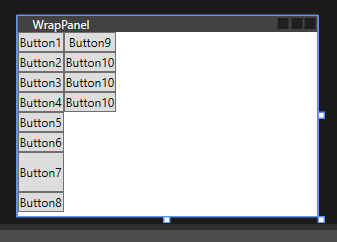
FruitListItem.Items.Add(childitem);

}

}

}

**WrapPanel:**

****

**Notes:**

* Wrap Panel is type of container like grid.
* Height matters only in Horizontally not Vertically & Width matters only in Vertically not Horizontally.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.WrapPanel"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="WrapPanel" Height="200" Width="300">

<WrapPanel Orientation="Vertical">

<Button>Button1</Button>

<Button>Button2</Button>

<Button>Button3</Button>

<Button>Button4</Button>

<Button>Button5</Button>

<Button>Button6</Button>

<Button Height="40">Button7</Button>

<Button>Button8</Button>

<Button>Button9</Button>

<Button>Button10</Button>

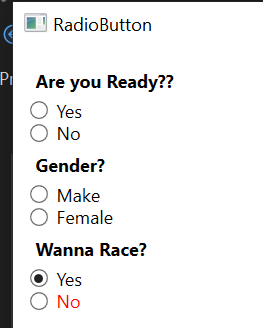
<Button>Button10</Button>

<Button>Button10</Button>

</WrapPanel>

</Window>

**Radio Button:**



**Notes:**

* Users can select from group one to one using Radio Button.
* If IsChecked is true then it is selected by default.

Ex- <RadioButton IsChecked="True">Yes</RadioButton>

* GroupName is used for selecting more than 1 choice in another checkbox.
* We can wrap radio button too using WrapPanel.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.RadioButton"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="RadioButton" Height="450" Width="800">

<StackPanel Margin="10">

<Label FontWeight="Bold">Are you Ready??</Label>

<RadioButton GroupName="A">Yes</RadioButton>

<RadioButton GroupName="A">No</RadioButton>

<Label FontWeight="Bold">Gender?</Label>

<RadioButton GroupName="B">Make</RadioButton>

<RadioButton GroupName="B">Female</RadioButton>

<Label FontWeight="Bold">Wanna Race?</Label>

<RadioButton IsChecked="True" GroupName="A">Yes</RadioButton>

<RadioButton GroupName="A">

<WrapPanel>

<TextBlock Text="No" Foreground="Red"></TextBlock>

</WrapPanel>

</RadioButton>

</StackPanel>

</Window>

**Password Box/Control:**

A screenshot of a computer screen

Description automatically generated

**Notes:**

* We have to use PasswordBox.
* PasswordChar=”x” means instead of DOT(.), x will show in PasswordBox.
* MaxLength=”8” means password cannot be more than 8.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.PasswordControl"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="PasswordControl" Height="450" Width="800">

<StackPanel>

<Label>Text:</Label>

<TextBox/>

<Label>Password:</Label>

<PasswordBox/>

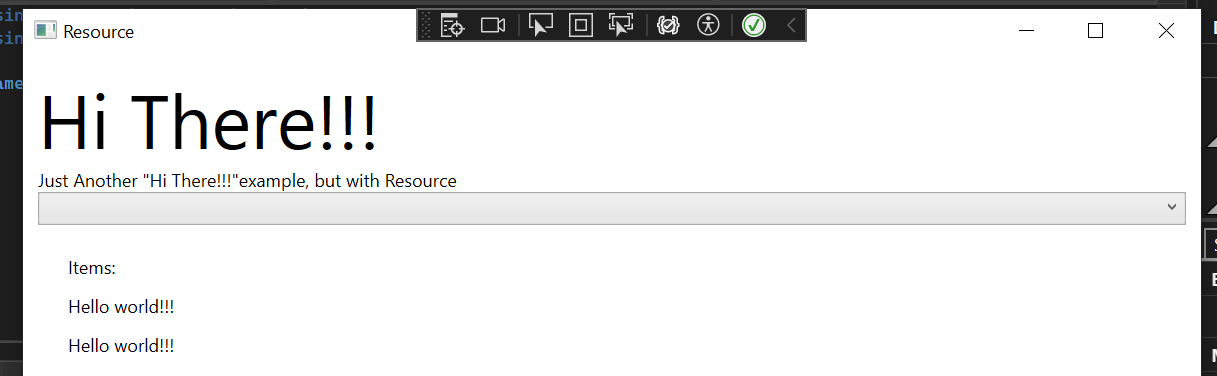
<Label>Another Password Example:</Label>

<PasswordBox PasswordChar="x" MaxLength="8"/>

</StackPanel>

</Window>

**Resources - Window, Local(MainWindow.xaml), Global(App.xaml):**

****

**Notes:**

* Sys is a key value pair.
* Add “xmlns:sys="clr-namespace:System;assembly=netstandard"” in namespace for sys error.
* Static Resource cannot be change but Dynamic can.
* Local Resource is used in MainWindow.xaml and Global Resource is used in App.xaml.
* In App.xaml add namespace too.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.Resource"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

xmlns:sys="clr-namespace:System;assembly=netstandard"

mc:Ignorable="d"

Title="Resource" Height="450" Width="800">

<Window.Resources>

<sys:String x:Key="SAK">Hi There!!!</sys:String>

<x:Array x:Key="Combo1" Type="sys:String">

<sys:String>First Item</sys:String>

<sys:String>Second Item</sys:String>

<sys:String>Third Item</sys:String>

</x:Array>

</Window.Resources>

<StackPanel Margin="10">

<TextBlock Text="{StaticResource SAK}" FontSize="50"/>

<TextBlock>Just Another "<TextBlock Text="{StaticResource SAK}"/>"example, but with Resource</TextBlock>

<ComboBox ItemsSource="{StaticResource Combo1}"/>

<StackPanel Margin="15">

<StackPanel.Resources>

<sys:String x:Key="Combo2">Items:</sys:String>

</StackPanel.Resources>

<Label Content="{StaticResource Combo2}"/>

<Label Content="{StaticResource strApp}"/>

<Label x:Name="l1"/>

</StackPanel>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

public partial class Resource : Window

{

public Resource()

{

InitializeComponent();

l1.Content = Application.Current.FindResource("strApp").ToString();

}

}

}

**In App.xaml->**

<Application x:Class="KfcApp2.App"

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

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

xmlns:local="clr-namespace:KfcApp2"

xmlns:sys="clr-namespace:System;assembly=netstandard"

StartupUri="Resource.xaml">

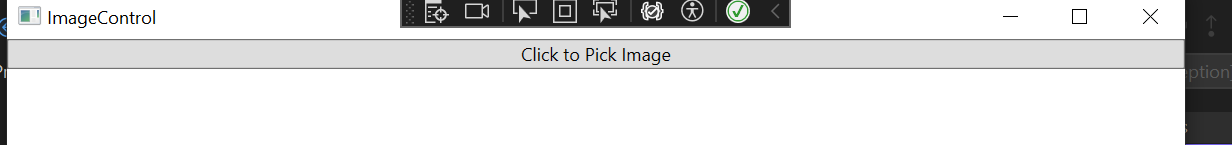
<Application.Resources>

<sys:String x:Key="strApp">Hello world!!!</sys:String>

</Application.Resources>

</Application>

**Image Control with example:**

****

**A screenshot of a computer

Description automatically generated**

**Notes:**

* For Image to be display, Left click on Image -> Properties -> Build Action = Resource.
* Click="Button\_Click" is event handling which will automatically creates a method in MainWindow.xaml.cs file.
* Instead of None, we can use uniform, fill etc. in stretch.

**Example1: For showing image->**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ImageControl"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="ImageControl" Height="450" Width="800">

<StackPanel>

<Image Source="/Images/kfclogo.jfif" Height="100" Width="100"/>

</StackPanel>

</Window>

**Example2: For Picking image from file->**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ImageControl"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="ImageControl" Height="450" Width="800">

<StackPanel>

<Button Content="Click to Pick Image" Click="Button\_Click"/>

<!--<Image Source="/Images/kfclogo.jfif" Height="100" Width="100"/>-->

<!--<Image x:Name="img1" Height="100" Width="100"/>-->

<!--<Image Source="/Images/kfclogo.jfif" x:Name="img1" Stretch="Uniform"/>-->

<Image x:Name="img1" Stretch="None"/>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

public partial class ImageControl : Window

{

public ImageControl()

{

InitializeComponent();

}

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

{

OpenFileDialog dialog = new OpenFileDialog();

if(dialog.ShowDialog()==true)

{

Uri fileuri = new Uri(dialog.FileName);

img1.Source = new BitmapImage(fileuri);

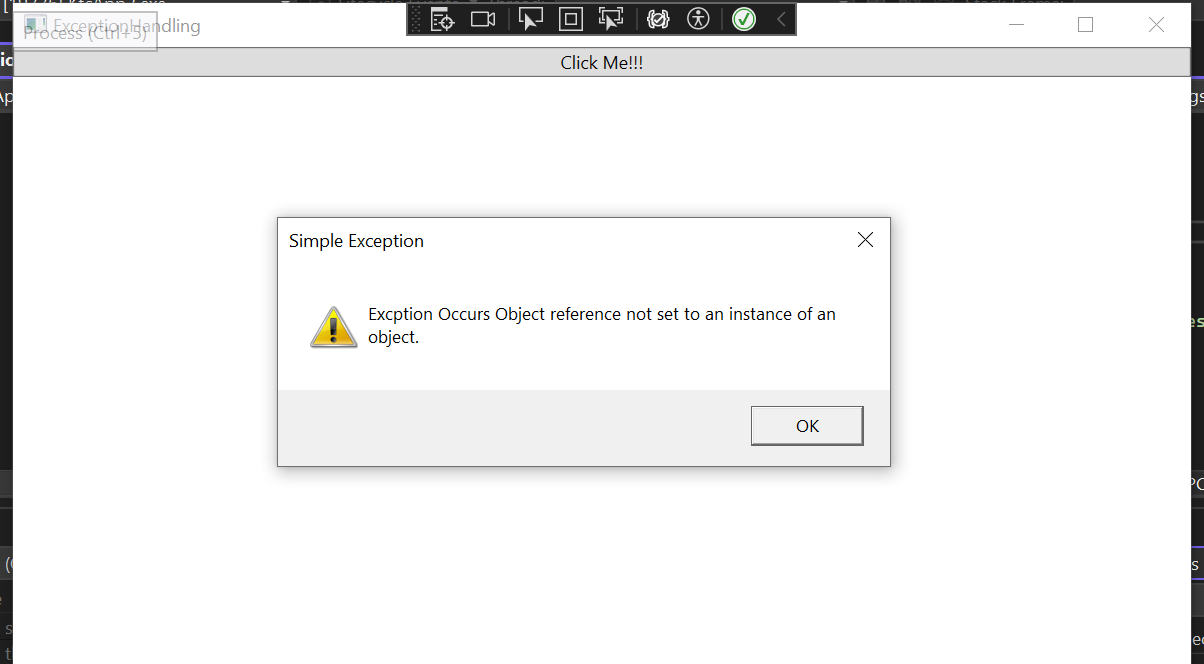
}

}

}

}

**Exception Handling:**

****

**Notes:**

* **Exception** is defined as an event that occurs during the execution of a program that is unexpected by the program code.
* We can use Try & Catch block for Exception Handling.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ExceptionHandling"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="ExceptionHandling" Height="450" Width="800">

<StackPanel>

<Button Content="Click Me!!!" Click="Button\_Click"></Button>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for ExceptionHandling.xaml

/// </summary>

public partial class ExceptionHandling : Window

{

public ExceptionHandling()

{

InitializeComponent();

}

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

{

string s = null;

try

{

s.Trim();

}

catch(Exception ex)

{

MessageBox.Show("Exception Occurs " + ex.Message, "Simple Exception", MessageBoxButton.OK, MessageBoxImage.Exclamation);

}

}

}

}

**CultureInfo – Application Culture/UI Culture:**

**A screenshot of a computer

Description automatically generated**

**Notes:**

* If someone wants to read the application in different language, then **CultureInfo** makes it easy to read the application.
* Use ”using System.Globalization;“ namespace for CultureInfo.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.CultureChangeInfo"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="CultureChangeInfo" Height="450" Width="800">

<StackPanel Margin="25">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label>Number:</Label>

<Label Name="lblNumber" Grid.Column="1"></Label>

<Label Grid.Row="1">Date:</Label>

<Label Name="lblDate" Grid.Row="1" Grid.Column="1"></Label>

</Grid>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="0,20">

<Button Tag="en-US" Click="Button\_Click" HorizontalContentAlignment="Stretch">English</Button>

<Button Tag="de-DE" Click="Button\_Click" HorizontalContentAlignment="Stretch" Margin="15,0">German</Button>

<Button Tag="sv-SE" Click="Button\_Click" HorizontalContentAlignment="Stretch">Sweden</Button>

</StackPanel>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.Globalization;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for CultureChangeInfo.xaml

/// </summary>

public partial class CultureChangeInfo : Window

{

public CultureChangeInfo()

{

InitializeComponent();

}

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

{

Thread.CurrentThread.CurrentCulture = new CultureInfo((sender as Button).Tag.ToString());

lblNumber.Content = (567897432.42d).ToString("N2");

lblDate.Content = DateTime.Now.ToString();

}

}

}

**Dialog Box- Save File, Font Dialog, Color Dialog:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Notes:**

* Dialog boxes are windows but with a specific intent and user experience.
* Dialog boxes are used to:

1. Display specific information to users.
2. Gather information from users.
3. Both display and gather information.
4. Display an operating system prompt, such as print window.
5. Select a file or folder.

* dialog.Filter = "Text File(\*.txt) | \*.txt | C# File(\*.cs) | \*.cs";   is used for providing extensions for “Save As”.
* using System.IO; for File Handling namespace.
* For **Font Dialog Box** we have to use “using System.Windows.Forms” namespace. For that we have to go to Tools -> Nugget Package Manager -> Search “Simplify.Windows.Forms” and install it.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DialogBox"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DialogBox" Height="450" Width="800">

<StackPanel>

<TextBox Text="" x:Name="t1" HorizontalAlignment="Center" VerticalAlignment="Center" Height="20" Width="100"/>

<Button Content="Save File" Click="Button\_Click" Height="20" Width="65" Margin="5"/>

<Button Content="Change Font" Click="Button\_Click\_1" Width="85"/>

<Button Content="Change Color" Click="Button\_Click\_2" Margin="5" Width="95"/>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Forms;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for DialogBox.xaml

/// </summary>

public partial class DialogBox : Window

{

public DialogBox()

{

InitializeComponent();

}

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

{

Microsoft.Win32.SaveFileDialog dialog = new Microsoft.Win32.SaveFileDialog();

dialog.Filter = "Text File(\*.txt) | \*.txt | C# File(\*.cs) | \*.cs";

dialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

if (dialog.ShowDialog() == true)

{

File.WriteAllText(dialog.FileName, t1.Text);

}

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

FontDialog dialog = new FontDialog();

if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

t1.FontFamily = new FontFamily(dialog.Font.Name);

t1.FontSize = dialog.Font.Size \* 90.0 / 72.0;

t1.FontWeight = dialog.Font.Bold ? FontWeights.Bold : FontWeights.Regular;

t1.FontStyle = dialog.Font.Italic ? FontStyles.Italic : FontStyles.Normal;

}

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

ColorDialog dialog = new ColorDialog();

if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

t1.Foreground = new SolidColorBrush(Color.FromArgb(dialog.Color.A,

dialog.Color.R, dialog.Color.G, dialog.Color.B));

}

}

}

}

**Panels – Canvas, Wrap Panel, Stack Panel, Dock Panel:**

**Notes:**

* Panels is used for designing (structural view)
* Wrap Panel = Horizontally
* Stack Panel = Vertically

**Canvas Panel:**

**A screen shot of a computer

Description automatically generated**

* **Canvas** is a **lightweight layout of WPF**. Canvas is used for 2D graphic design elements but not for UI. You cannot use it for making text-box, checkbox, or drop-down because it will create difficulty in screen resolution. Canvas has its height & width, so it cannot be re-sized automatically. It allows you to put canvas into the canvas.
* Ellipse = Circle
* Panel.ZIndex="5" is used to show in uppermost layer.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.CanvasEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="CanvasEx" Height="450" Width="800">

<Canvas>

<Button Canvas.Left="25">First Button</Button>

<Button Canvas.Right="25">Second Button</Button>

<Button Canvas.Bottom="25" Canvas.Left="25">Third Button</Button>

<Button Canvas.Bottom="25" Canvas.Right="25">Fourth Button</Button>

<Ellipse Panel.ZIndex="2" Fill="Aqua" Canvas.Left="25" Canvas.Top="25" Width="200" Height="200"/>

<Rectangle Panel.ZIndex="3" Fill="Aquamarine" Canvas.Left="25" Canvas.Top="25" Width="50" Height="50"/>

<Rectangle Panel.ZIndex="5" Fill="LightCoral" Canvas.Left="50" Canvas.Top="50" Width="50" Height="50"/>

<Rectangle Panel.ZIndex="4" Fill="LightGray" Canvas.Left="75" Canvas.Top="75" Width="50" Height="50"/>

</Canvas>

</Window>

**WrapPanel:**

**Go To Page 21**

**StackPanel:**

**A screenshot of a computer

Description automatically generated**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.StackPanel"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="StackPanel" Height="450" Width="800">

<StackPanel>

<Button HorizontalAlignment="Left">Button 1</Button>

<Button HorizontalAlignment="Right">Button 2</Button>

<Button>Button 3</Button>

<Button>Button 4</Button>

<Button>Button 5</Button>

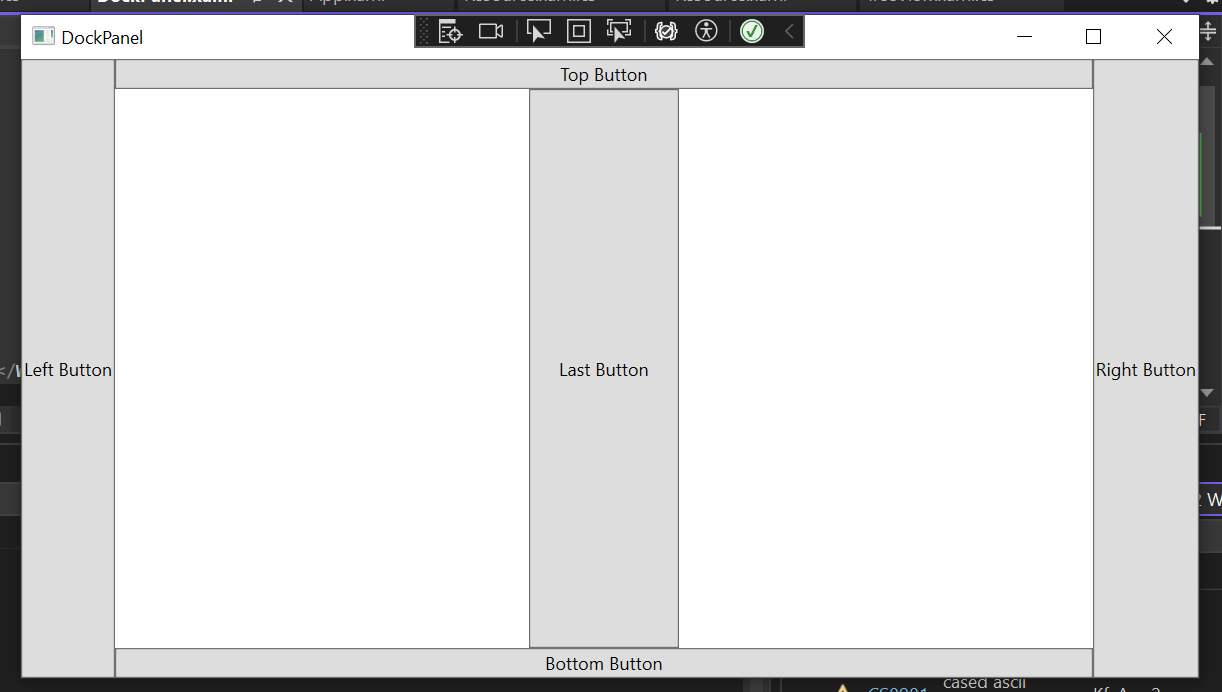
<Button >Button 6</Button>

</StackPanel>

</Window>

**DockPanel:**

* DockPanel works in directions i.e left, right, top, bottom.



**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DockPanel"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DockPanel" Height="450" Width="800">

<DockPanel>

<Button DockPanel.Dock="Left">Left Button</Button>

<Button DockPanel.Dock="Right">Right Button</Button>

<Button DockPanel.Dock="Top">Top Button</Button>

<Button DockPanel.Dock="Bottom">Bottom Button</Button>

<Button Width="100">Last Button</Button>

</DockPanel>

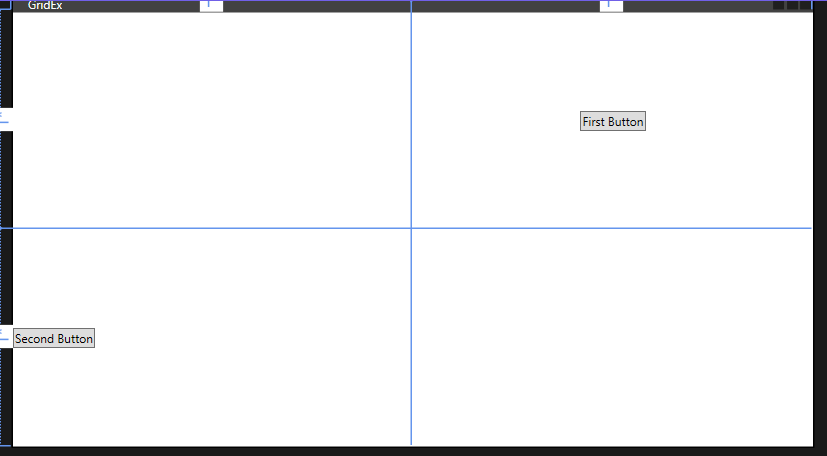
</Window>

**Grid – Row and Columns, Grid Units, Grid Spanning:**

**Notes:**

* Grid has rows and columns by default.

**Grid:**



**In MainWindow.xaml->**

<Window x:Class="KfcApp2.GridEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="GridEx" Height="450" Width="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Button Grid.Row="0" Grid.Column="1" Content="First Button" HorizontalAlignment="Center" VerticalAlignment="Center"/>

<Button Grid.Row="1" HorizontalAlignment="Left" VerticalAlignment="Center">Second Button</Button>

</Grid>

</Window>

**Grid Units:**

**A screenshot of a computer

Description automatically generated**

* Width="\*" means which ever column takes more space, size of the entire column will be that much.
* Width="Auto" means which it will takes the size of the Content.
* Width="100" means 100 will be the fixed size of the column.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.GridUnits"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="GridUnits" Height="450" Width="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="150"/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Button Content="First Button" Grid.Column="0"/>

<Button Content="Second Button" Grid.Column="1"/>

<Button Content="Third Button for Clicking Purpose" Grid.Column="2"/>

<Button Content="Forth Button" Grid.Column="3"/>

</Grid>

</Window>

**Grid Spanning:**

**A screenshot of a computer

Description automatically generated**

* <Button Grid.ColumnSpan="2">Button 1</Button> means Button 1 will take first two columns.
* <Button Grid.Column="3">Button 2</Button> means Button 2 will be in Row0 and Column3.
* <Button Grid.Row="1">Button 3</Button> means Button 3 will be in Row1 and Column0.
* <Button Grid.Column="1" Grid.Row="1" Grid.RowSpan="2" Grid.ColumnSpan="2">Button 4</Button> means Button 4 will start from Row1 and Column1 and ends from Row2 and Column2.
* <Button Grid.Column="0" Grid.Row="2">Button 5</Button> means Button 5 will takes place Row2 and Column0.

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.GridSpanning"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="GridSpanning" Height="450" Width="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Button Grid.ColumnSpan="2">Button 1</Button>

<Button Grid.Column="3">Button 2</Button>

<Button Grid.Row="1">Button 3</Button>

<Button Grid.Column="1" Grid.Row="1" Grid.RowSpan="2" Grid.ColumnSpan="2">Button 4</Button>

<Button Grid.Column="0" Grid.Row="2">Button 5</Button>

</Grid>

</Window>

**User Control:**

**Notes:**

* **Step 1:** Right Click on Project in Solution Explorer -> Add -> **User Control (WPF)** -> UserControlLoginPage.xaml

**Step 2:** Right Click on Project in Solution Explorer -> Add -> **Window (WPF)** -> UserControlConsume.xaml

* **For reusability we use User Control.**
* this.DataContext = this; means the page is bind within themselves.
* By default, Orientation is vertical, so we have to take Horizontal.
* Element Name = Text Box Name.
* In UserControlConsume.xaml we have to use namespace like xmlns:local="clr-namespace:KfcApp2"

**UserControlLoginPage.xaml** (which we can use it again and again):A screenshot of a computer

Description automatically generated

**UserControlConsume.xaml** (which is the main Window Page and taken from User Control):

A screenshot of a computer

Description automatically generated

**In UserControlLoginPage.xaml->**

<UserControl x:Class="KfcApp2.UserControlLoginPage"

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

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

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label Grid.Row="0" Grid.Column="0" Content="{Binding Title}"/>

<Label Grid.Row="0" Grid.Column="1">

<StackPanel Orientation="Horizontal">

<TextBlock Text="{Binding ElementName=first, Path=Text.Length}"/>

<TextBlock Text="/" />

<TextBlock Text="{Binding MaxLength}"/>

</StackPanel>

</Label>

<TextBox MaxLength="{Binding MaxLength}" x:Name="first" TextWrapping="Wrap" Grid.Row="1" Grid.ColumnSpan="2" Width="400" Height="150"/>

</Grid>

</UserControl>

**In UserControlLoginPage.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for UserControlLoginPage.xaml

/// </summary>

public partial class UserControlLoginPage : UserControl

{

public UserControlLoginPage()

{

InitializeComponent();

this.DataContext = this;

}

public string Title { get; set; }

public string MaxLength { get; set; }

}

}

**In UserControlConsume.xaml->**

<Window x:Class="KfcApp2.UserControlConsume"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="UserControlConsume" Height="450" Width="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<local:UserControlLoginPage Title="Enter Title:" MaxLength="50" Grid.Row="0"/>

<local:UserControlLoginPage Title="Enter Description:" MaxLength="300" Grid.Row="1"/>

</Grid>

</Window>

**Data Binding (Basic):**

**Notes:**

* Binding – Synchronization between two sources.
* UpdateSourceTrigger=PropertyChanged means effect going to change in real time.

**Example 1: Basic Data Binding without Effect in Real Time**

**A screenshot of a computer

Description automatically generated**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DataBindingDemo"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DataBindingDemo" Height="450" Width="800">

<StackPanel>

<TextBox x:Name="First"/>

<TextBlock Text="{Binding Path=Text,ElementName=First}"/>

</StackPanel>

</Window>

**Example 2: Update in real time**

**A screenshot of a computer

Description automatically generated**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DataBindingDemo"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DataBindingDemo" Height="450" Width="800">

<StackPanel>

<TextBox Text="{Binding Height, UpdateSourceTrigger=PropertyChanged}"/>

<TextBox Text="{Binding Width, UpdateSourceTrigger=PropertyChanged}"/>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System.Windows;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for DataBindingDemo.xaml

/// </summary>

public partial class DataBindingDemo : Window

{

public DataBindingDemo()

{

InitializeComponent();

this.DataContext = this;

}

}

}

**Example 3: Update Height after Clicking Button**

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DataBindingDemo"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DataBindingDemo" Height="450" Width="800">

<StackPanel>

<TextBox x:Name="HeightTxt" Text="{Binding Height, UpdateSourceTrigger=Explicit}"/>

<TextBox Text="{Binding Width, UpdateSourceTrigger=PropertyChanged}"/>

<Button Content="Click to change Height" Click="Button\_Click" Height="25" Width="150"/>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for DataBindingDemo.xaml

/// </summary>

public partial class DataBindingDemo : Window

{

public DataBindingDemo()

{

InitializeComponent();

this.DataContext = this;

}

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

{

BindingExpression exp = HeightTxt.GetBindingExpression(TextBox.TextProperty);

exp.UpdateSource();

}

}

}

**Data Binding via Code Behind – INotifyPropertyChanged, Observable Collection:**

**Notes:**

* The **INotifyPropertyChanged** interface is used to notify clients, typically binding clients, that a property value has changed[**1**](https://www.bing.com/ck/a?!&&p=fcc500983d9bfba7JmltdHM9MTcxMzMxMjAwMCZpZ3VpZD0yMDY4YjUyYS0yMDZlLTY3NjItMmIzMC1hNjljMjFhZjY2Y2YmaW5zaWQ9NTUwNg&ptn=3&ver=2&hsh=3&fclid=2068b52a-206e-6762-2b30-a69c21af66cf&psq=what+is+inotifypropertychanged+in+c%23&u=a1aHR0cHM6Ly9sZWFybi5taWNyb3NvZnQuY29tL2VuLXVzL2RvdG5ldC9hcGkvc3lzdGVtLmNvbXBvbmVudG1vZGVsLmlub3RpZnlwcm9wZXJ0eWNoYW5nZWQ_dmlldz1uZXQtNy4w&ntb=1). This interface is commonly implemented by classes that are used as data sources for data binding.
* An **ObservableCollection** is a dynamic collection of objects of a given type. Objects can be added, removed or be updated with an automatic notification of actions. When an object is added to or removed from an observable collection, the UI is automatically updated. This happens because, when binding to an observable collection, WPF automatically adds a [CollectionChanged](https://docs.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection-1.collectionchanged?redirectedfrom=MSDN&view=netframework-4.8) event handler to the ObservableCollecion's events.

The **ObservableCollection** class exists in the **System.Collections.ObjectModel** namespace.

**Example 1: Data Binding without code behind**

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DataBindingAdvanced"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DataBindingAdvanced" Height="450" Width="800">

<StackPanel>

<TextBox x:Name="nametxt"/>

<TextBlock x:Name="resultblock" Text="{Binding Path=Text, ElementName=nametxt}"/>

</StackPanel>

</Window>

**Example 2: Code Behind Data Binding**

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DataBindingAdvanced"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DataBindingAdvanced" Height="450" Width="800">

<StackPanel>

<TextBox x:Name="nametxt"/>

<TextBlock x:Name="resultblock"/>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for DataBindingAdvanced.xaml

/// </summary>

public partial class DataBindingAdvanced : Window

{

public DataBindingAdvanced()

{

InitializeComponent();

Binding b1 = new Binding("Text");

b1.Source = nametxt;

resultblock.SetBinding(TextBlock.TextProperty, b1);

}

}

}

**Example 3: Bind ListView with List of Data by taking Observable Collection and INotifyPropertyChange.**

**Notes:**

* We have to add Plugin for **INotifyPropertyChange** and for that: Right on the project name -> Manage Nuget Package -> Search “**PropertyChange.Fody”** and install it.
* using System.Collections.ObjectModel;
* using PropertyChanged;
* [AddINotifyPropertyChangedInterface]
* An **ObservableCollection** is a dynamic collection of objects of a given type. Objects can be added, removed or be updated with an automatic notification of actions. When an object is added to or removed from an observable collection, the UI is automatically updated. This happens because, when binding to an observable collection, WPF automatically adds a **[CollectionChanged](https://docs.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection-1.collectionchanged?redirectedfrom=MSDN&view=netframework-4.8)** event handler to the ObservableCollecion's events.

The ObservableCollection class exists in the **System.Collections.ObjectModel** namespace.

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.DataBindingListView"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="DataBindingListView" Height="450" Width="800">

<DockPanel Margin="10">

<StackPanel DockPanel.Dock="Right" Margin="10,0,0,0">

<Button Content="Add Product" Click="Button\_Click" Margin="2"/>

<Button Content="Change Product" Click="Button\_Click\_1" Margin="2"/>

<Button Content="Delete Product" Click="Button\_Click\_2" Margin="2"/>

</StackPanel>

<ListBox x:Name="list1" DisplayMemberPath="Name"/>

</DockPanel>

</Window>

**In MainWindow.xaml.cs->**

using PropertyChanged;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for DataBindingListView.xaml

/// </summary>

public partial class DataBindingListView : Window

{

ObservableCollection<Product> products = new ObservableCollection<Product>();

public DataBindingListView()

{

InitializeComponent();

//List of Data

products.Add(new Product { Name = "Camera" });

products.Add(new Product { Name = "TV" });

products.Add(new Product { Name = "Mobiile" });

products.Add(new Product { Name = "Radio" });

list1.ItemsSource = products;

}

[AddINotifyPropertyChangedInterface]

public class Product

{

public string Name { get; set; }

}

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

{

//To Add

products.Add(new Product { Name = "Watch" });

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

if (list1.SelectedItem != null)

{

//To Change

(list1.SelectedItem as Product).Name = "Electronic Gadget";

}

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

if (list1.SelectedItem != null)

{

//To Delete

products.Remove(list1.SelectedItem as Product);

}

}

}

}

**Value Converter Using IValueConverter, StringFormat Property:**

**Notes:**

* If you want to **databind** two properties that have incompatible types, you need a piece of code in between, that converts the value from source to target type and back. This piece of code is called **ValueConverter**. A **value converter** is a class, that implements the simple interface IValueConverter with the two methods object Convert(object value) and object ConvertBack(object value).
* After writing “public class ChangeValue : IValueConverter”, we have to implement **IValueConverter** by right clicking on it and selecting **Implement IValueConverter** then we will get **Convert** and **ConvertBack** function.
* **Resources** - The ability to store data as a resource, either locally for a control, locally for the entire window or globally for the entire application. The data can be pretty much whatever you want, from actual information to a hierarchy of WPF controls. This allows you to place data in one place and then use it from or several other places, which is very useful.
* We have to make **resources/Content:**

<Window.Resources>

<local:ChangeValue x:Key="r1"/>

</Window.Resources>

* We have to add “ xmlns:system="clr-namespace:System;assembly=netstandard" ” namespace for Static system.

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ValueConverter"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

xmlns:system="clr-namespace:System;assembly=netstandard"

mc:Ignorable="d"

Title="ValueConverter" Height="450" Width="800">

<Window.Resources>

<local:ChangeValue x:Key="r1"/>

</Window.Resources>

<StackPanel>

<TextBox x:Name="v1"/>

<CheckBox Content="Yes" IsChecked="{Binding Path=Text, ElementName=v1, Converter={StaticResource r1}}"/>

<StackPanel>

<TextBlock Text="{Binding Source = {x:Static system:DateTime.Now},StringFormat=Date:{0:dd/mm/yyyy}}"/>

</StackPanel>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

public partial class ValueConverter : Window

{

public ValueConverter()

{

InitializeComponent();

}

}

public class ChangeValue : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if(value.ToString().ToLower() == "yes")

{

return true;

}

return false;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

if((bool)value==true)

{

return "yes";

}

return false;

}

}

}

**Dialog Box: Message Box Dialog, Open File Dialog, Save File Dialog**

**Notes:**

* **MessageBox** class has an overloaded static Show method that displays a message box with a message and action buttons. The action buttons can be OK and Cancel, Yes and No etc.
* **OpenFileDialog** control allows us to browse and select files on a computer in an application. A typical Open File Dialog looks where you can see Windows Explorer like features to navigate through folders and select a file.
* Use “using Microsoft.Win32;” for OpenFileDialog.
* **SaveFileDialog** control is used to save a file using Windows SaveFileDialog. A typical SaveFileDialog looks like Figure 1 where you can see the Windows Explorer type features to navigate through folders and save a file in a folder.

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.MessageBoxEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="MessageBoxEx" Height="450" Width="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<StackPanel Grid.Row="0" Orientation="Horizontal">

<Button Content="Open Message Box" Click="Button\_Click" Width="120" Height="25"/>

<Button Content="Open File Box" Click="OpenDialogBoxEvent" Width="120" Height="25" Margin="20"/>

<Button Content="Save File Box" Click="SaveDialogBoxEvent" Width="120" Height="25" Margin="0"/>

</StackPanel>

<StackPanel Grid.Row="1">

<TextBox x:Name="ResultOpen" VerticalScrollBarVisibility="Visible" AcceptsReturn="True" TextWrapping="Wrap"/>

</StackPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for MessageBoxEx.xaml

/// </summary>

public partial class MessageBoxEx : Window

{

public MessageBoxEx()

{

InitializeComponent();

}

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

{

MessageBoxResult result = MessageBox.Show("Hello World", "MessageBox Example", MessageBoxButton.YesNoCancel, MessageBoxImage.Information, MessageBoxResult.No);

switch(result)

{

case (MessageBoxResult.Yes):MessageBox.Show("Click Yes");

break;

case (MessageBoxResult.No):

MessageBox.Show("Click No");

break;

case (MessageBoxResult.Cancel):

MessageBox.Show("Click Cancel");

break;

}

}

private void OpenDialogBoxEvent(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "png File (\*.png)|\*.png | All Files (\*.\*)|\*.\*";

openFileDialog.Multiselect = true;

openFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

if(openFileDialog.ShowDialog()==true)

{

ResultOpen.Text = File.ReadAllText(openFileDialog.FileName);

}

}

private void SaveDialogBoxEvent(object sender, RoutedEventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Text File(\*.txt) | \*.txt | png File (\*.png)|\*.png | All Files (\*.\*)|\*.\*";

saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);

if (saveFileDialog.ShowDialog() == true)

{

File.WriteAllText(saveFileDialog.FileName, ResultOpen.Text);

}

}

}

}

**Toolbar, Toolbar Tray: Overflow, Image Icon with text**

**Notes:**

* A ToolBar control is a group of Toolbar button controls that can be clicked to achieve a functionality, forexample the toolbars in Microsoft Word and Visual Studio where you see File Open, Save and Print buttons.
* CanExecute is a Event Handling.
* New Command is in Overflow Mode.
* <Separator/> is line in between Toolbar.

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ToolBarExample"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="ToolBarExample" Height="450" Width="800">

<Window.CommandBindings>

<CommandBinding Command="New" CanExecute="CommandBinding\_CanExecute"/>

<CommandBinding Command="Open" CanExecute="CommandBinding\_CanExecute"/>

<CommandBinding Command="Save" CanExecute="CommandBinding\_CanExecute"/>

<CommandBinding Command="Cut" CanExecute="CommandBinding\_CanExecute"/>

<CommandBinding Command="Copy" CanExecute="CommandBinding\_CanExecute"/>

<CommandBinding Command="Paste" CanExecute="CommandBinding\_CanExecute"/>

</Window.CommandBindings>

<Grid>

<DockPanel>

<ToolBarTray DockPanel.Dock="Top">

<ToolBar>

<Button Command="New" Content="New" ToolBar.OverflowMode="Always"/>

<Button Command="Open" Content="Open"/>

<Button Command="Save" Content="Save"/>

</ToolBar>

<ToolBar>

<Button Command="Cut">

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\cutimg.jfif" Width="20" Height="20"/>

</Button>

<Button Command="Copy" Content="Copy"/>

<Button Command="Paste">

<StackPanel Orientation="Horizontal">

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\Pasteimg.jfif" Height="20" Width="20"/>

<TextBlock Text="Paste" Margin="4,0,0,0"/>

</StackPanel>

</Button>

<Separator/>

<Label>Font Families</Label>

<ComboBox>

<ComboBoxItem IsSelected="True">Ariel</ComboBoxItem>

<ComboBoxItem>Roman</ComboBoxItem>

<ComboBoxItem>Italic</ComboBoxItem>

</ComboBox>

</ToolBar>

</ToolBarTray>

</DockPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for ToolBarExample.xaml

/// </summary>

public partial class ToolBarExample : Window

{

public ToolBarExample()

{

InitializeComponent();

}

private void CommandBinding\_CanExecute(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = true;

}

}

}

**Status Bar:**

**Notes:**

* A StatusBar is a horizontal window that usually sits at the bottom of a window to display various kinds of status information of an application.
* A screenshot of a computer

  Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.StatusBarExample"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="StatusBarExample" Height="450" Width="800">

<Grid>

<DockPanel>

<StatusBar DockPanel.Dock="Bottom">

<!--<StatusBarItem>

<TextBlock x:Name="txt1"/>

</StatusBarItem>-->

<StatusBar.ItemsPanel>

<ItemsPanelTemplate>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="200"/>

<ColumnDefinition Width="20"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

</Grid>

</ItemsPanelTemplate>

</StatusBar.ItemsPanel>

<StatusBarItem>

<TextBlock x:Name="txt1"/>

</StatusBarItem>

<Separator Grid.Column="1"/>

<StatusBarItem Grid.Column="2">

<TextBlock Text="c:\MyDocument\file.txt"/>

</StatusBarItem>

</StatusBar>

<TextBox AcceptsReturn="True" x:Name="inputtxt" TextChanged="inputtxt\_TextChanged" Height="100" Width="200"/>

</DockPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for StatusBarExample.xaml

/// </summary>

public partial class StatusBarExample : Window

{

public StatusBarExample()

{

InitializeComponent();

}

private void inputtxt\_TextChanged(object sender, TextChangedEventArgs e)

{

int row = inputtxt.GetLineIndexFromCharacterIndex(inputtxt.CaretIndex);

int col = inputtxt.CaretIndex - inputtxt.GetLineIndexFromCharacterIndex(row);

txt1.Text = "Line No. " + (row + 1) + " Characters: " + (col + 1);

}

}

}

**Border Control:**

**A screenshot of a computer

Description automatically generated**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.BorderExample"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="BorderExample" Height="450" Width="800">

<Grid Margin="30">

<!--<Border Background="LightBlue" BorderBrush="Red" BorderThickness="6"

CornerRadius="30,0,30,0">-->

<Border BorderBrush="Red" BorderThickness="6"

CornerRadius="30,0,30,0">

<Border.Background>

<LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">

<GradientStop Color="LightBlue" Offset="0.0"/>

<GradientStop Color="LightGreen" Offset="0.5"/>

<GradientStop Color="DarkRed" Offset="1.0"/>

</LinearGradientBrush>

</Border.Background>

<StackPanel Margin="15">

<Button>First</Button>

<Button Margin="0,20">Second</Button>

<Button>Third</Button>

</StackPanel>

</Border>

</Grid>

</Window>

**Slider Control:**

**A screenshot of a computer

Description automatically generated**

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.SliderExample"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="SliderExample" Height="450" Width="800">

<StackPanel>

<StackPanel Margin="10" VerticalAlignment="Center">

<TextBlock Text="{Binding ElementName=slider1,Path=Value,UpdateSourceTrigger=PropertyChanged}"/>

<Slider ValueChanged="slider1\_ValueChanged" x:Name="slider1" Maximum="255" TickPlacement="BottomRight" TickFrequency="10" IsSnapToTickEnabled="True"/>

</StackPanel>

<StackPanel Margin="10" VerticalAlignment="Center">

<TextBlock Text="{Binding ElementName=slider2,Path=Value,UpdateSourceTrigger=PropertyChanged}"/>

<Slider ValueChanged="slider1\_ValueChanged" x:Name="slider2" Maximum="255" TickPlacement="BottomRight" TickFrequency="10" IsSnapToTickEnabled="True"/>

</StackPanel>

<StackPanel Margin="10" VerticalAlignment="Center">

<TextBlock Text="{Binding ElementName=slider3,Path=Value,UpdateSourceTrigger=PropertyChanged}"/>

<Slider ValueChanged="slider1\_ValueChanged" x:Name="slider3" Maximum="255" TickPlacement="BottomRight" TickFrequency="10" IsSnapToTickEnabled="True"/>

</StackPanel>

</StackPanel>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for SliderExample.xaml

/// </summary>

public partial class SliderExample : Window

{

public SliderExample()

{

InitializeComponent();

}

private void slider1\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

Color color = Color.FromRgb((byte)slider1.Value, (byte)slider2.Value, (byte)slider3.Value);

this.Background = new SolidColorBrush(color);

}

}

}

**Progress Bar:**

**Notes:**

* ContentRendered="Window\_ContentRendered" is a event handler.
* using System.ComponentModel; is required for BackgroundWorker.
* IsIndeterminate="True" means progress bar will continuously run i.e

<ProgressBar x:Name="Status1" Minimum="0" Maximum="100" Value="20" IsIndeterminate="True">

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.ProgressBarEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d" ContentRendered="Window\_ContentRendered"

Title="ProgressBarEx" Height="450" Width="800">

<Grid>

<StackPanel Margin="40">

<Grid>

<ProgressBar x:Name="Status1" Minimum="0" Maximum="100"/>

<TextBlock Text="{Binding ElementName=Status1, Path=Value, StringFormat={}{0:0}%}"/>

</Grid>

</StackPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for ProgressBarEx.xaml

/// </summary>

public partial class ProgressBarEx : Window

{

public ProgressBarEx()

{

InitializeComponent();

}

private void Window\_ContentRendered(object sender, EventArgs e)

{

//for (int i = 0; i < 100; i++)

//{

// Status1.Value++;

// Thread.Sleep(100);

//}

BackgroundWorker backworker = new BackgroundWorker();

backworker.WorkerReportsProgress = true;

backworker.DoWork += backworker\_DoWork;

backworker.ProgressChanged += backworker\_ProgressChanged;

backworker.RunWorkerAsync();

}

private void backworker\_ProgressChanged(object? sender, ProgressChangedEventArgs e)

{

Status1.Value = e.ProgressPercentage;

}

private void backworker\_DoWork(object? sender, DoWorkEventArgs e)

{

for (int i = 0; i <= 100; i++)

{

(sender as BackgroundWorker).ReportProgress(i);

Thread.Sleep(100);

}

}

}

}

**Web Browser:**

**Notes:**

* <Window.CommandBindings></Window.CommandBindings> has predefined commands.

A screenshot of a computer

Description automatically generated

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.WebBrowserEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="WebBrowserEx" Height="450" Width="800">

<Window.CommandBindings>

<CommandBinding Command="NavigationCommands.BrowseBack" CanExecute="CommandBinding\_CanExecute" Executed="CommandBinding\_Executed"/>

<CommandBinding Command="NavigationCommands.BrowseForward" CanExecute="CommandBinding\_CanExecute\_1" Executed="CommandBinding\_Executed\_1"/>

<CommandBinding Command="NavigationCommands.GoToPage" CanExecute="CommandBinding\_CanExecute\_2" Executed="CommandBinding\_Executed\_2"/>

</Window.CommandBindings>

<Grid>

<DockPanel>

<ToolBar DockPanel.Dock="Top">

<Button Command="NavigationCommands.BrowseBack">

<Image Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\Back.jfif" Height="22" Width="22"/>

</Button>

<Button Command="NavigationCommands.BrowseForward">

<Image Source="https://creazilla-store.fra1.digitaloceanspaces.com/cliparts/7813969/rightwards-arrow-clipart-md.png" Height="20" Width="20"/>

</Button>

<Separator/>

<TextBox x:Name="urltxt" Width="600" TextWrapping="Wrap" KeyUp="urltxt\_KeyUp"/>

<Button Command="NavigationCommands.GoToPage">

Go

</Button>

</ToolBar>

<WebBrowser x:Name="WebBroswer1" Navigating="WebBroswer1\_Navigating">

</WebBrowser>

</DockPanel>

</Grid>

</Window>

**In MainWindow.xaml.cs->**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace KfcApp2

{

/// <summary>

/// Interaction logic for WebBrowserEx.xaml

/// </summary>

public partial class WebBrowserEx : Window

{

public WebBrowserEx()

{

InitializeComponent();

WebBroswer1.Navigate("https://www.google.co.in/");

}

private void WebBroswer1\_Navigating(object sender, System.Windows.Navigation.NavigatingCancelEventArgs e)

{

//MessageBox.Show(e.Uri.OriginalString);

urltxt.Text = e.Uri.OriginalString;

}

private void CommandBinding\_CanExecute(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = ((WebBroswer1 != null) && (WebBroswer1.CanGoBack));

}

private void CommandBinding\_CanExecute\_1(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = ((WebBroswer1 != null) && (WebBroswer1.CanGoForward));

}

private void CommandBinding\_CanExecute\_2(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = true;

}

private void CommandBinding\_Executed(object sender, ExecutedRoutedEventArgs e)

{

WebBroswer1.GoBack();

}

private void CommandBinding\_Executed\_1(object sender, ExecutedRoutedEventArgs e)

{

WebBroswer1.GoForward();

}

private void CommandBinding\_Executed\_2(object sender, ExecutedRoutedEventArgs e)

{

WebBroswer1.Navigate(urltxt.Text);

}

private void urltxt\_KeyUp(object sender, KeyEventArgs e)

{

if(e.Key == Key.Enter)

{

WebBroswer1.Navigate(urltxt.Text);

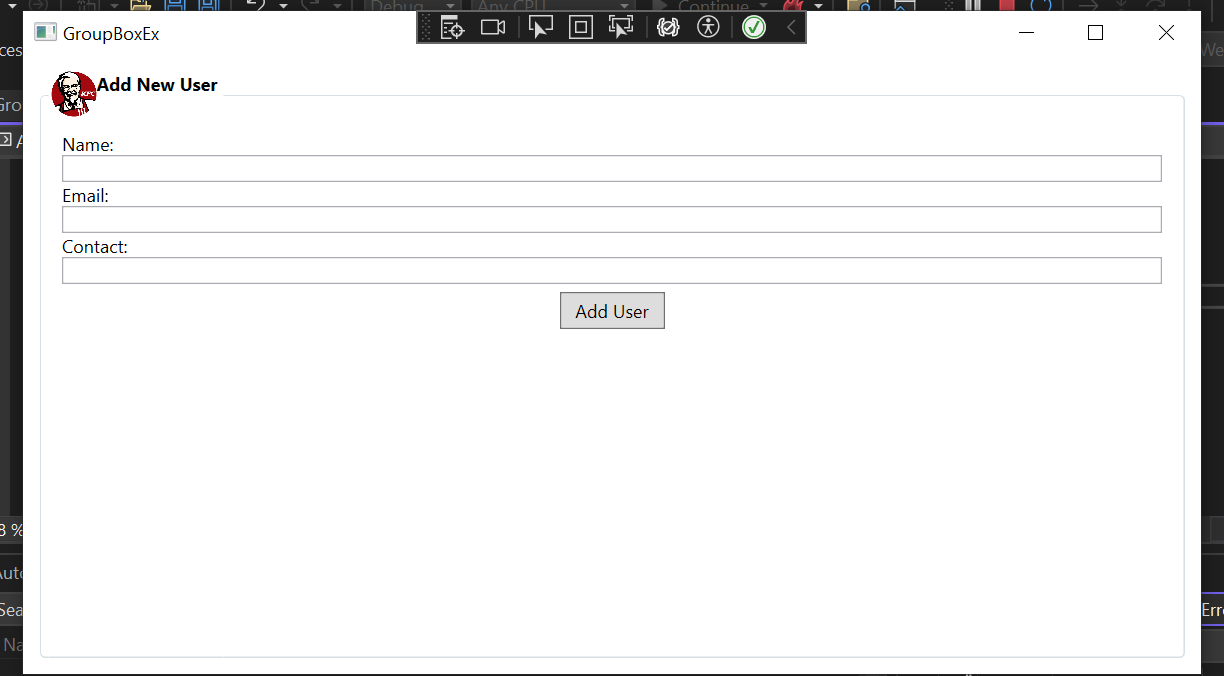
}

}

}

}

**GroupBox:**



**In MainWindow.xaml->**

<Window x:Class="KfcApp2.GroupBoxEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="GroupBoxEx" Height="450" Width="800">

<Grid>

<!--<GroupBox Header="Add New User" Margin="15" Padding="15">

<StackPanel>

<TextBlock>Name:</TextBlock>

<TextBox x:Name="nametxt"/>

<TextBlock>Email:</TextBlock>

<TextBox x:Name="emailtxt"/>

<TextBlock>Contact:</TextBlock>

<TextBox x:Name="contacttxt"/>

<Button Margin="5" Height="25" Width="70">Add User</Button>

</StackPanel>

</GroupBox>-->

<GroupBox Margin="10" Padding="10">

<GroupBox.Header>

<StackPanel Orientation="Horizontal">

<Image Height="30" Width="30" Source="C:\Users\sahid.khan2\source\repos\KfcApp2\Images\kfclogo.jfif"/>

<TextBlock FontWeight="Bold">Add New User</TextBlock>

</StackPanel>

</GroupBox.Header>

<StackPanel>

<TextBlock>Name:</TextBlock>

<TextBox x:Name="nametxt"/>

<TextBlock>Email:</TextBlock>

<TextBox x:Name="emailtxt"/>

<TextBlock>Contact:</TextBlock>

<TextBox x:Name="contacttxt"/>

<Button Margin="5" Height="25" Width="70">Add User</Button>

</StackPanel>

</GroupBox>

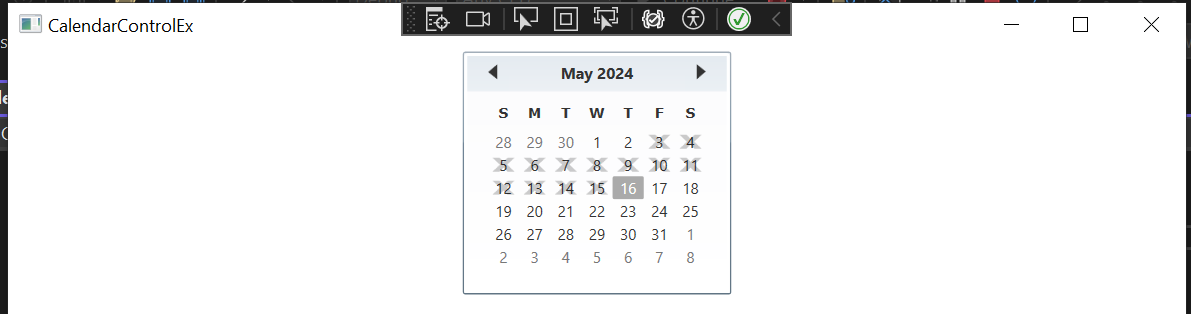
</Grid>

</Window>

**Calendar Control:**

**A screenshot of a calendar

Description automatically generated**

****

**In MainWindow.xaml->**

<Window x:Class="KfcApp2.CalendarControlEx"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:KfcApp2"

mc:Ignorable="d"

Title="CalendarControlEx" Height="450" Width="800">

<!--<Grid>

<Calendar/>

</Grid>-->

<!--<Viewbox Stretch="Fill" StretchDirection="UpOnly">

--><!--<Calendar DisplayDate="01.01.2023"/>-->

<!--<Calendar SelectionMode="MultipleRange"/>--><!--

</Viewbox>-->

<StackPanel>

<Calendar x:Name="cal1"/>

<Calendar x:Name="cal2" SelectionMode="MultipleRange"/>

<TextBox Text="{Binding ElementName=cal1, Path=SelectedDate}"/>

<ListBox ItemsSource="{Binding ElementName=cal2, Path=SelectedDates}" MinHeight="150"/>

</StackPanel>

<!--<StackPanel>

<Calendar DisplayMode="Year"/>

</StackPanel>--><!--

<StackPanel>

<Calendar SelectionMode="MultipleRange">

<Calendar.BlackoutDates>

<CalendarDateRange Start="05.03.2024" End="05.15.2024"/>

</Calendar.BlackoutDates>

</Calendar>

</StackPanel>-->

</Window>