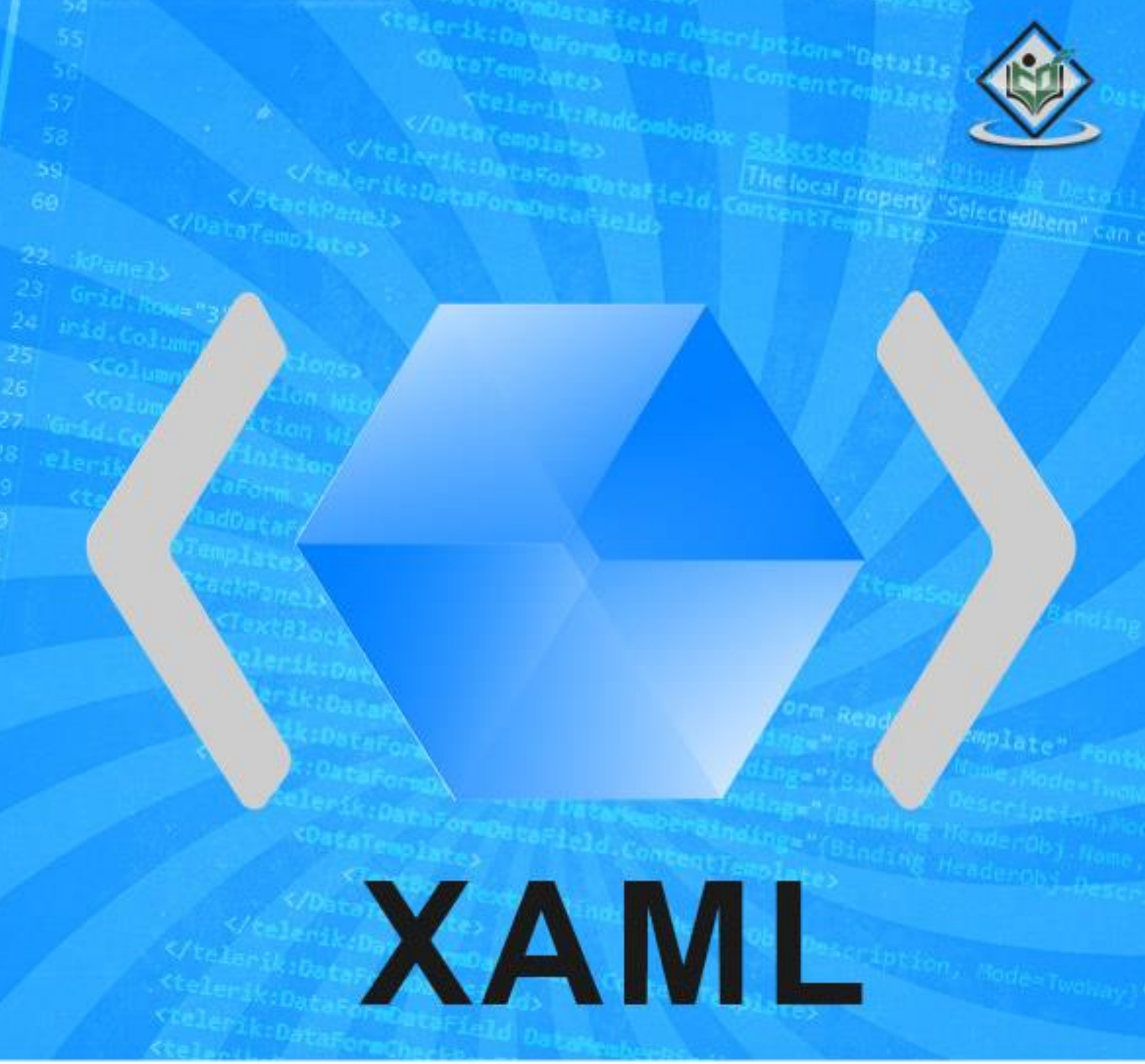


XAML

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

Welcome to the XAML tutorial for beginners. This tutorial puts greater emphasis on real-time implementation of the concept rather than discussing just the theory part.

The primary objective of this tutorial is to provide you a better understating of what you can do with XAML development irrespective of the platform you are using.

Audience

This tutorial has been designed for all those readers who want to learn XAML and to apply it instantaneously in different type of applications.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of XML, Web Technologies, and HTML.

Disclaimer & Copyright

Table of Contents

1. XAML – OVERVIEW

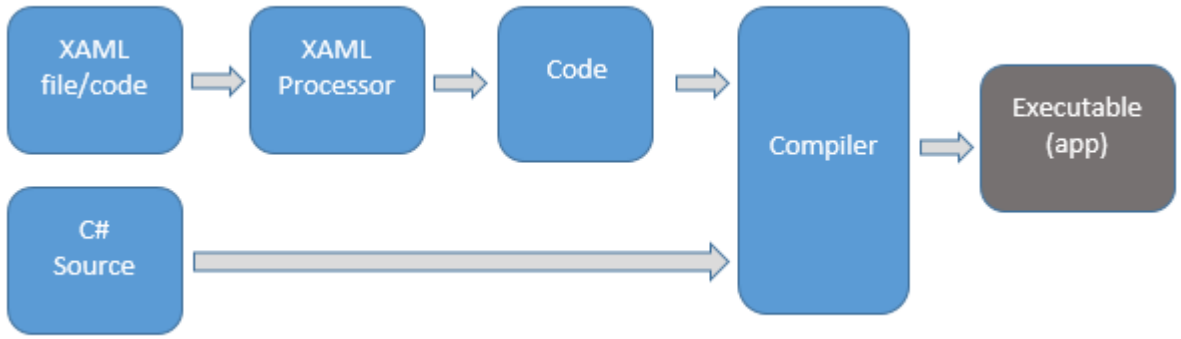
XAML stands for Extensible Application Markup Language. It's a simple and declarative language based on XML.

- In XAML, it very easy to create, initialize, and set properties of an object with hierarchical relations.
- It is mainly used for designing GUIs.
- It can be used for other purposes as well, e.g., to declare workflow in Workflow Foundation.

XAML can be used in different platforms such as WPF (Windows Presentation Foundation), Silverlight, Mobile Development, and Windows Store App. It can be used across different .Net framework and CLR (common language runtime) versions.

How XAML Works

XAML is a **declarative** language in the sense it defines the **WHAT** and **HOW** you want to do. XAML processor is responsible for the **HOW** part to find out. Let's have a look at the following schema. It sums up the XAML side of things:



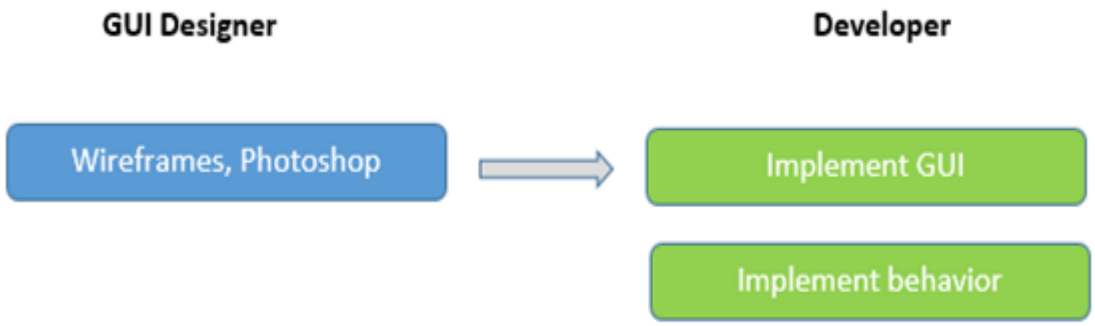
The figure illustrates the following actions:

- The XAML file is interpreted by a platform-specific XAML processor.
- The XAML processor transforms the XAML to internal code that describes the UI element.
- The internal code and the C# code are linked together through partial classes definitions and then the .NET compiler builds the app.

Advantages of XAML

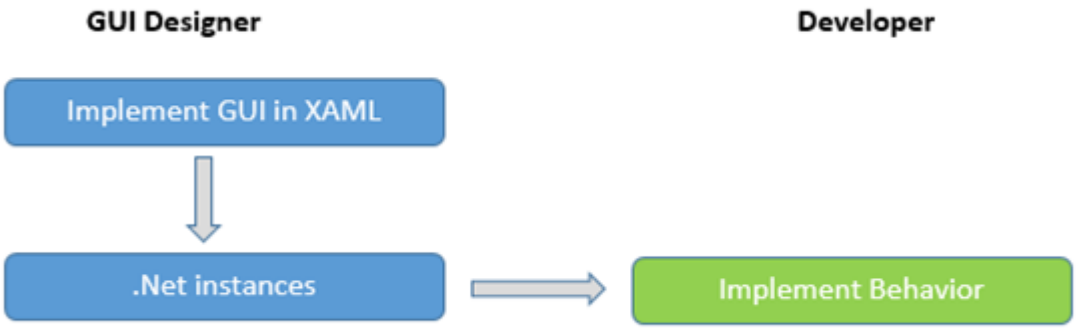
One of the longstanding problems that all of us face with GUI design can be solved by using XAML. It can be used to design UI elements in Windows Forms applications.

In the earlier GUI frameworks, there was no real separation between how an application looks like and how it behaves. Both the GUI and its behavior were created in the same language, e.g. C# or VB.net, which would require more effort from the developer to implement both the UI and the behavior associated with it.



Earlier GUI Frameworks

With XAML, it is very easy to separate the behavior from the designer code. Hence, the XAML programmer and the designer can work in parallel. XAML codes are very easy to read and understand.



XAML Framework

2. XAML – ENVIRONMENT SETUP

Microsoft provides two important tools for XAML:

- Visual Studio
- Expression Blend

Currently, both the tools can create XAML, but the fact is that Visual Studio is used more by developers while Expression Blend is still used more often by designers.

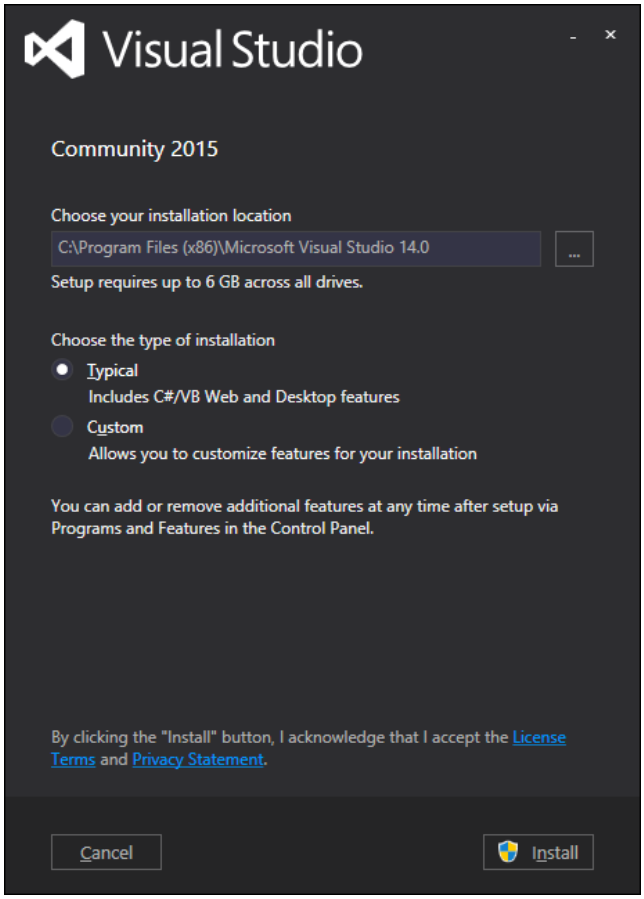
Microsoft provides a free version of Visual Studio which can be downloaded from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx

Note: For this tutorial, we will mostly be using WPF projects and Windows Store App. But the free version of Visual Studio doesn't support Windows Store App. So for that purpose, you will need a licensed version of Visual Studio.

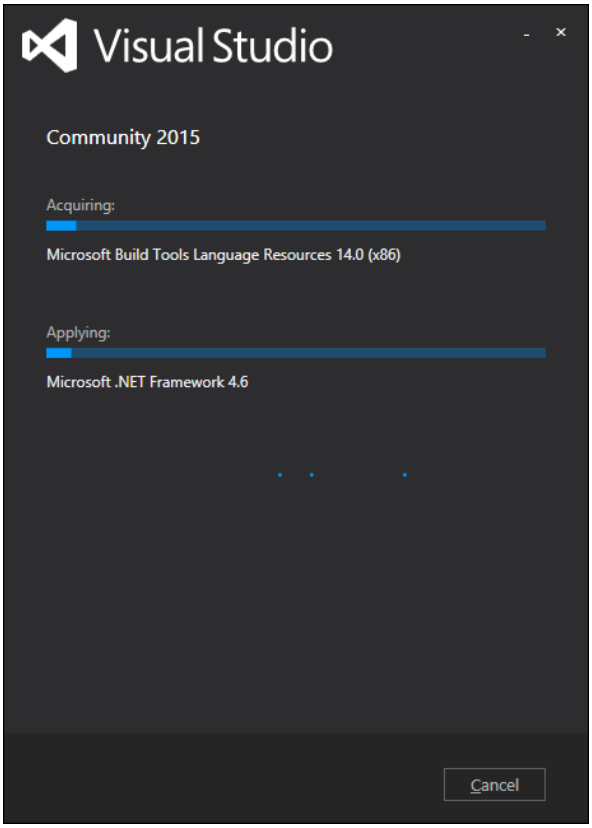
Installation

Follow the steps given below to install Visual Studio on your system:

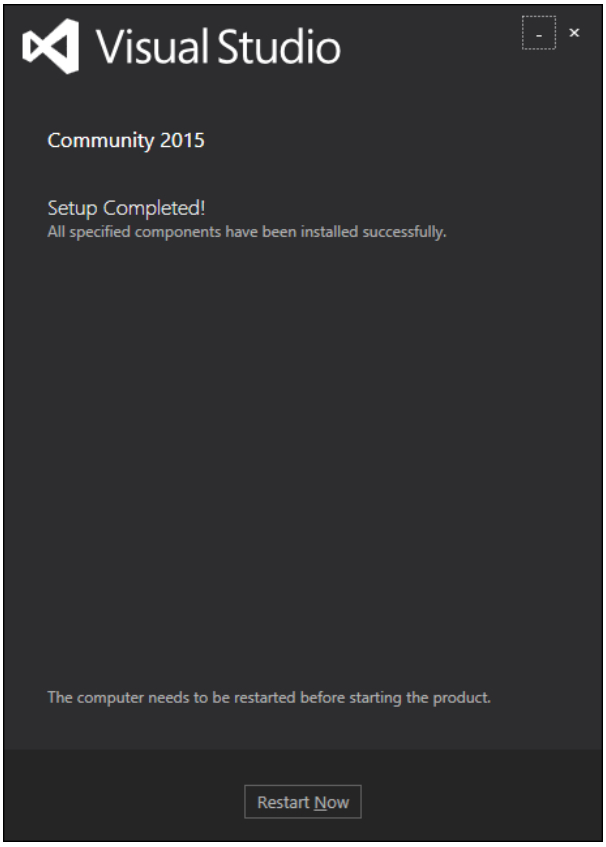
1. After downloading the files, run the installer. The following dialog box will be displayed.



2. Click on the Install button and it will start the installation process.

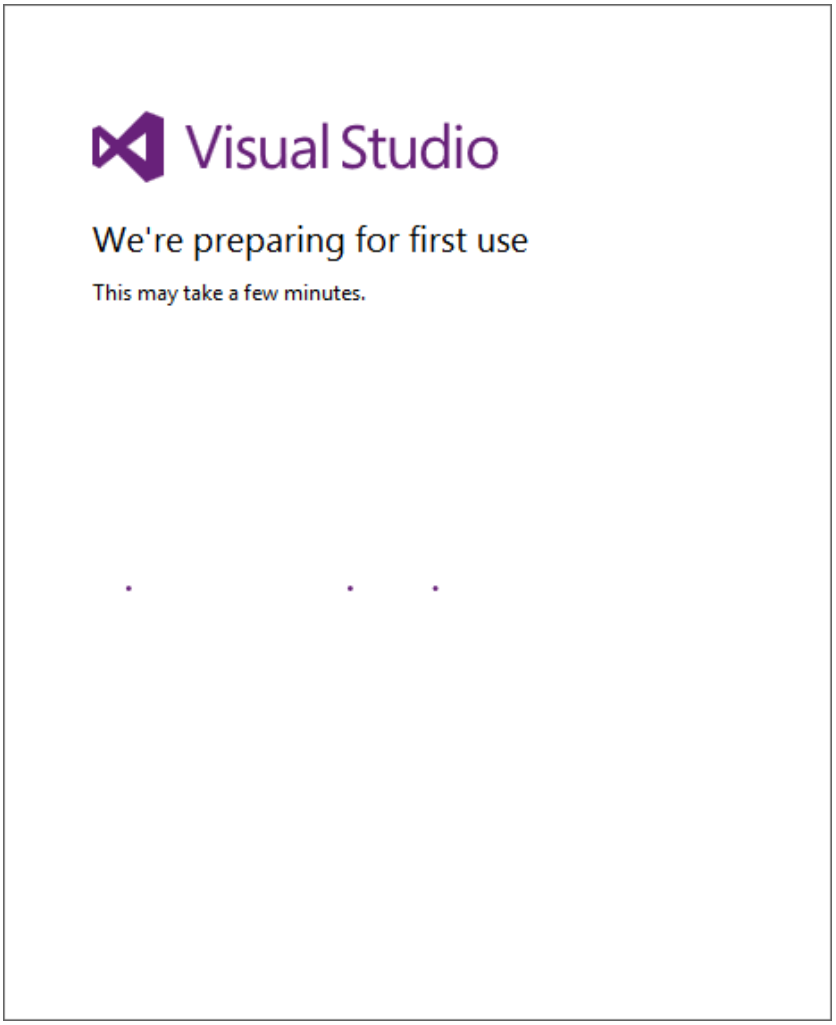


3. Once the installation process completes successfully, you will see the following screen.

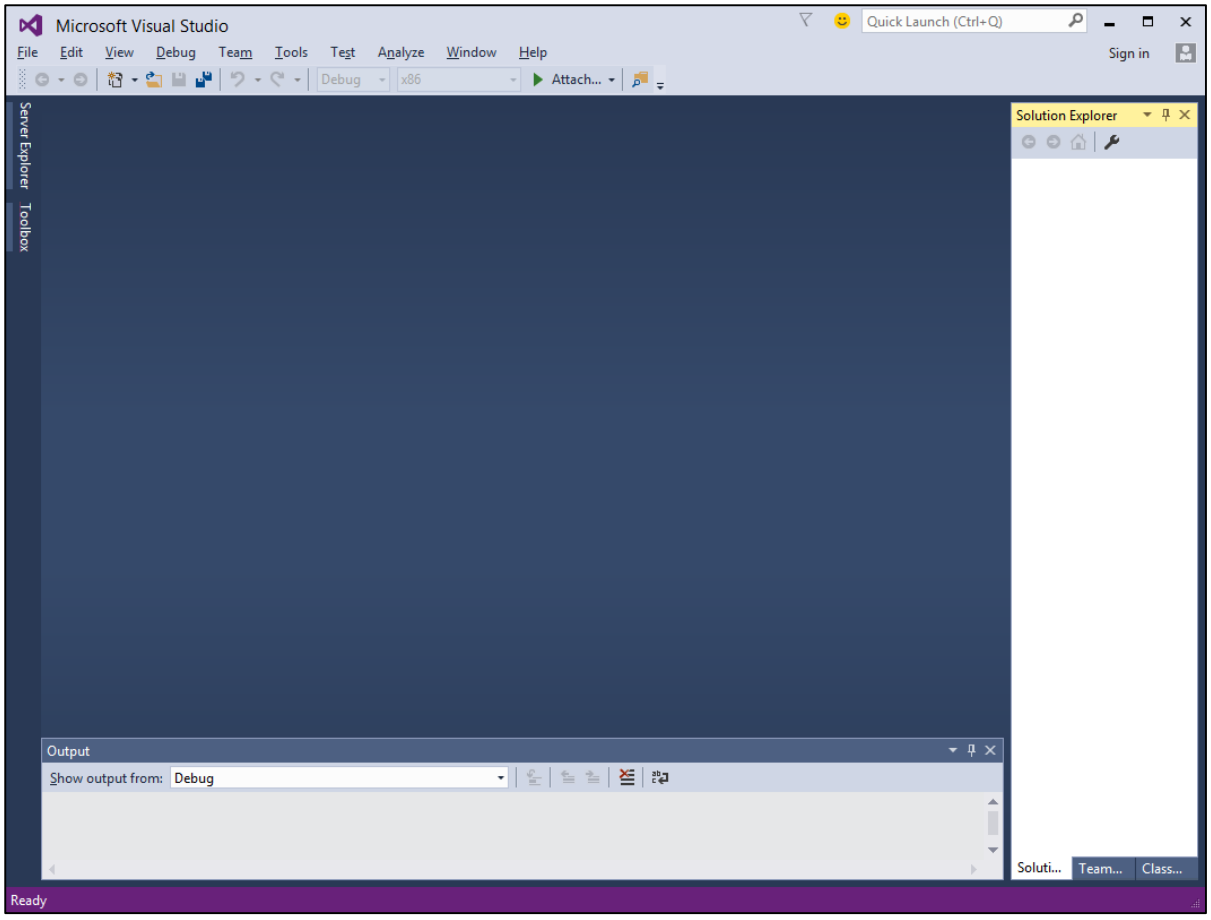


4. Close this dialog box and restart your computer if required.

5. Now open Visual studio from the Start Menu which will show the following dialog box. It will take some time for the first time, only for preparation.



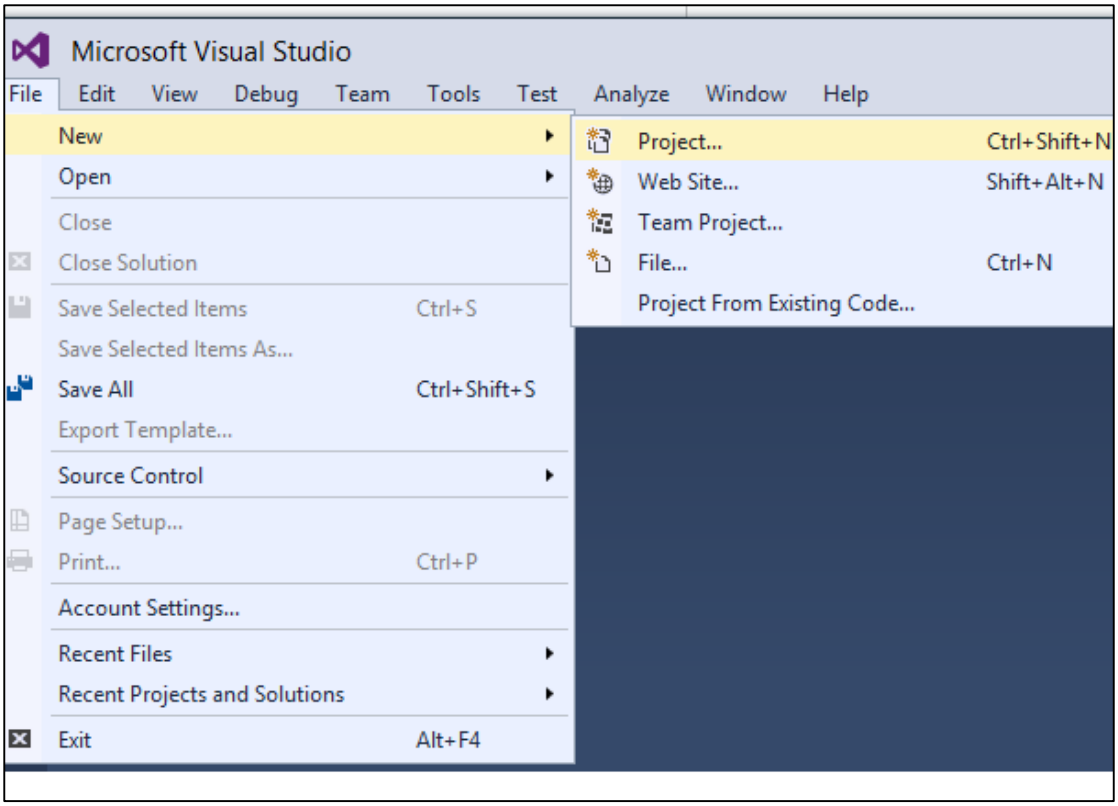
Once all is done, you will see the main window of Visual Studio.



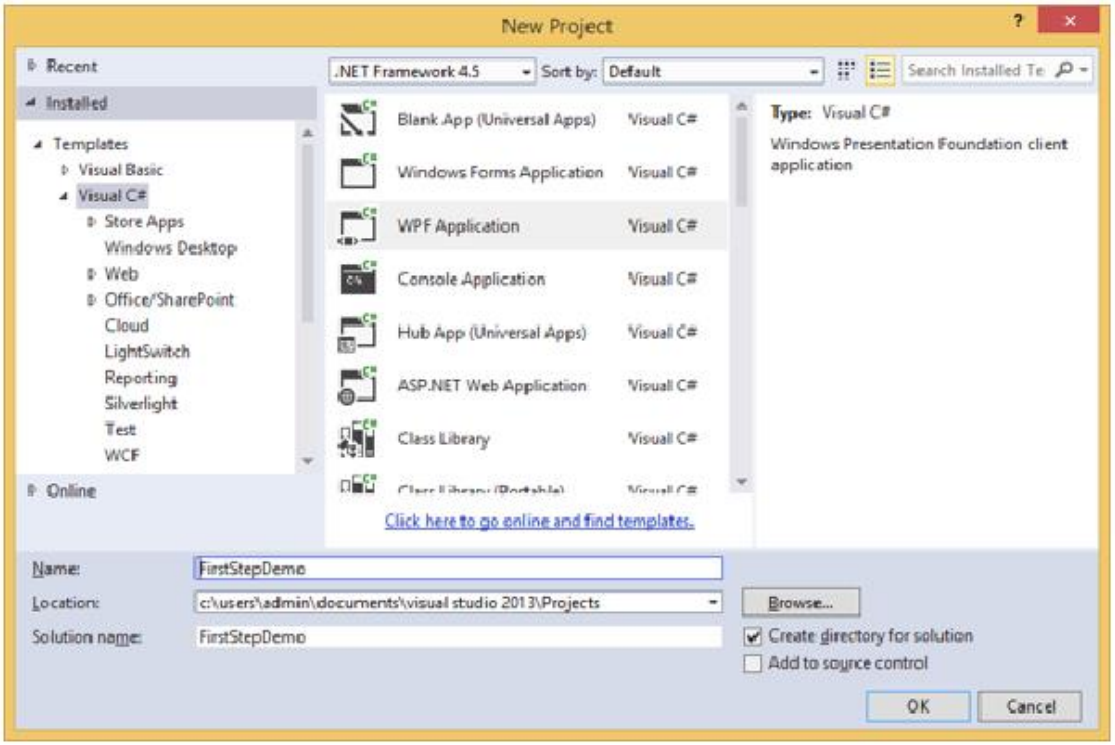
First Step towards Implementation

Let us start with a simple implementation. Follow the steps given below:

1. Click on File > New > Project menu option.



2. The following dialog box will be displayed:



3. Under Templates, select Visual C# and select WPF Application. Give a name to the project and click the OK button.

4. In the mainwindow.xaml file, the following XAML tags are written by default. You will understand all these tags later in this tutorial.

```
<Window x:Class="FirstStepDemo.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:FirstStepDemo"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="604">
    <Grid>

    </Grid>
</Window>
```

By default, a grid is set as the first element after page.

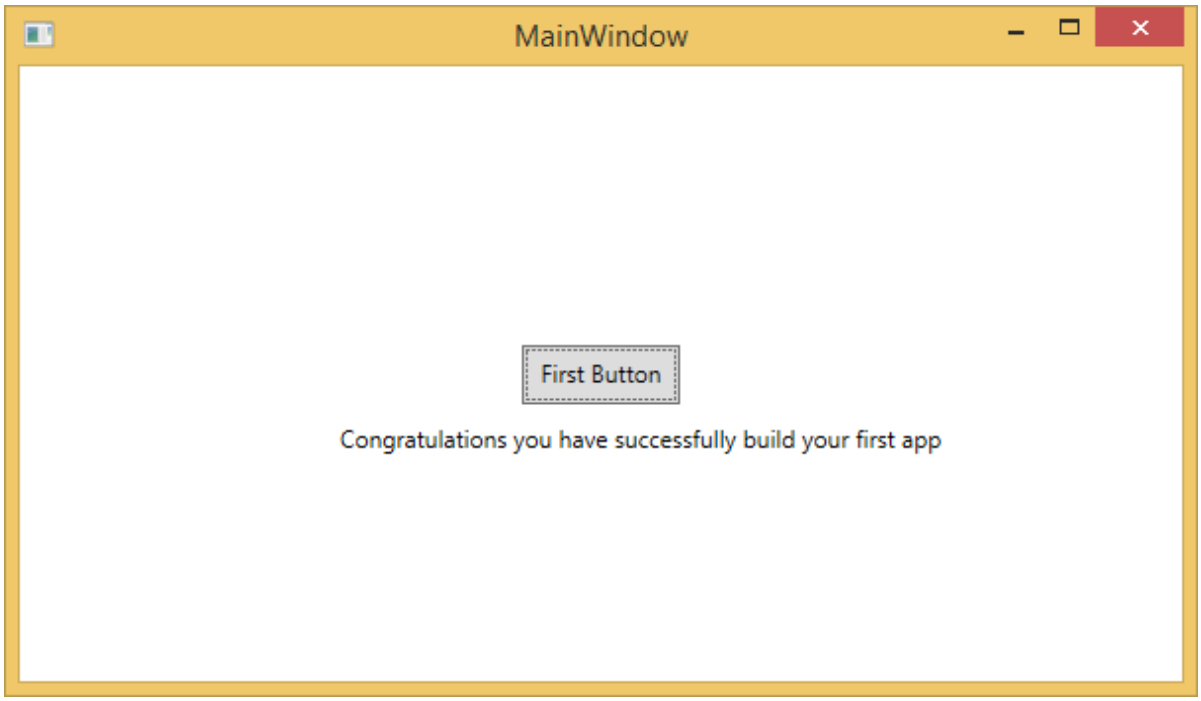
Let's add a button and a text block under the Grid element. This is called **object element syntax**, a left angle bracket followed by the name of what we want to instantiate, for example a button, then define a content property. The string assigned to the Content will be displayed on the button. Now set the height and width of the button as 30 and 50 respectively. Similarly initialize the properties of the Text block.

Now look at the design window. You will get to see a button. Now press F5 to execute this XAML code.

```
<Window x:Class="FirstStepDemo.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:FirstStepDemo"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="604">
    <Grid>
        <Button Content="First Button" Height="30" Width="80"/>
        <TextBlock Text="Congratulations you have successfully build your first
app"
                   Height="30" Margin="162,180,122,109" />
    </Grid>
```

```
</Window>
```

When you compile and execute the above code, you will see the following window.



Congratulation! You have designed your First Button.

3. WRITING XAML APPLICATION ON MAC OS

XAML applications can be developed on Mac as well. On Mac, XAML can be used as iOS and Android applications. To setup the environment on Mac, go to xamarin.com. Click on Products and select the Xamarin Platform. Download Xamarin Studio and install it. It will allow you to develop applications for the various platforms.

XAML – C# Syntax

In this chapter, you will learn the basic XAML syntax/rules to write XAML applications. Let's have a look at a simple XAML file.

```
<Window x:Class="Resources.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

As you can see in the above XAML file, there are different kinds of tags and elements. The following table briefly describes all the elements.

<Window	It is the opening object element or container of the root.
x:Class="Resources.MainWindow"	It is the partial class declaration which connects the markup to the partial class code behind defined in it.
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"	Maps the default XAML namespace for WPF client/framework
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"	XAML namespace for XAML language which maps it to x: prefix
>	End of object element of the root.
<Grid> </Grid>	Starting and closing tags of an empty grid object.

`</Window>`	Closing the object element

Syntax Rules for Object Element

Syntax rules for XAML is almost similar to XML. If you take a look at an XAML document, then you will notice that actually it is a valid XML file. However, an XML file cannot be a valid XAML file. It is because in XML, the value of the attributes must be a string, while in XAML, it can be a different object which is known as Property element syntax.

- The syntax of an Object element starts with a left angle bracket (<) followed by the name of the object, e.g. Button
- Define some Properties and attributes of that object element
- The Object element must be closed by a forward slash (/) followed immediately by a right angle bracket (>).

Example of simple object with no child element:

```
<Button/>
```

Example of object element with some attributes:

```
<Button Content="Click Me"
        Height="30"
        Width="60"/>
```

Example of an alternate syntax to define properties (Property element syntax):

```
<Button
        <Button.Content>Click Me</Button.Content>
        <Button.Height>30</Button.Height>
        <Button.Width>60</Button.Width>
</Button>
```

Example of Object with Child Element: StackPanel contains Textblock as child element

```
<StackPanel Orientation="Horizontal">
            <TextBlock Text="Hello"/>
</StackPanel>
```

4. XAML VS C# CODE

You can use XAML to create, initialize, and set the properties of objects. The same activities can also be performed using programming code.

XAML is just another simple and easy way to design UI elements. With XAML, it is up to you to decide whether you want to declare objects in XAML or declare them using code.

Let's take a simple example to demonstrate how to write in XAML:

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

    <StackPanel>
        <TextBlock Text="Welcome to XAML Tutorial" Height="20" Width="200"
                   Margin="5"/>
        <Button Content="Ok" Height="20" Width="60" Margin="5"/>
    </StackPanel>
</Window>
```

In this example, we have created a stack panel with a Button and a Text block and defined some of the properties of button and text block such as Height, Width, and Margin. When the above code is compiled and executed, it will produce the following output:

Now look at the same code which is written in C#.

```
using System;
using System.Text;
using System.Windows;
using System.Windows.Controls;

namespace XAMLVsCode
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            // Create the StackPanel
            StackPanel stackPanel = new StackPanel();
            this.Content = stackPanel;
```

```
            // Create the TextBlock
            TextBlock textBlock = new TextBlock();
            textBlock.Text = "Welcome to XAML Tutorial";
            textBlock.Height = 20;
            textBlock.Width = 200;
            textBlock.Margin = new Thickness(5);
            stackPanel.Children.Add(textBlock);

            // Create the Button
            Button button = new Button();
            button.Content = "OK";
            button.Height = 20;
            button.Width = 50;
            button.Margin = new Thickness(20);
            stackPanel.Children.Add(button);
        }
    }
}
```

When the above code is compiled and executed, it will produce the following output. Note that it is exactly the same as the output of XAML code.

Now you can see that how simple it is to use and understand XAML.

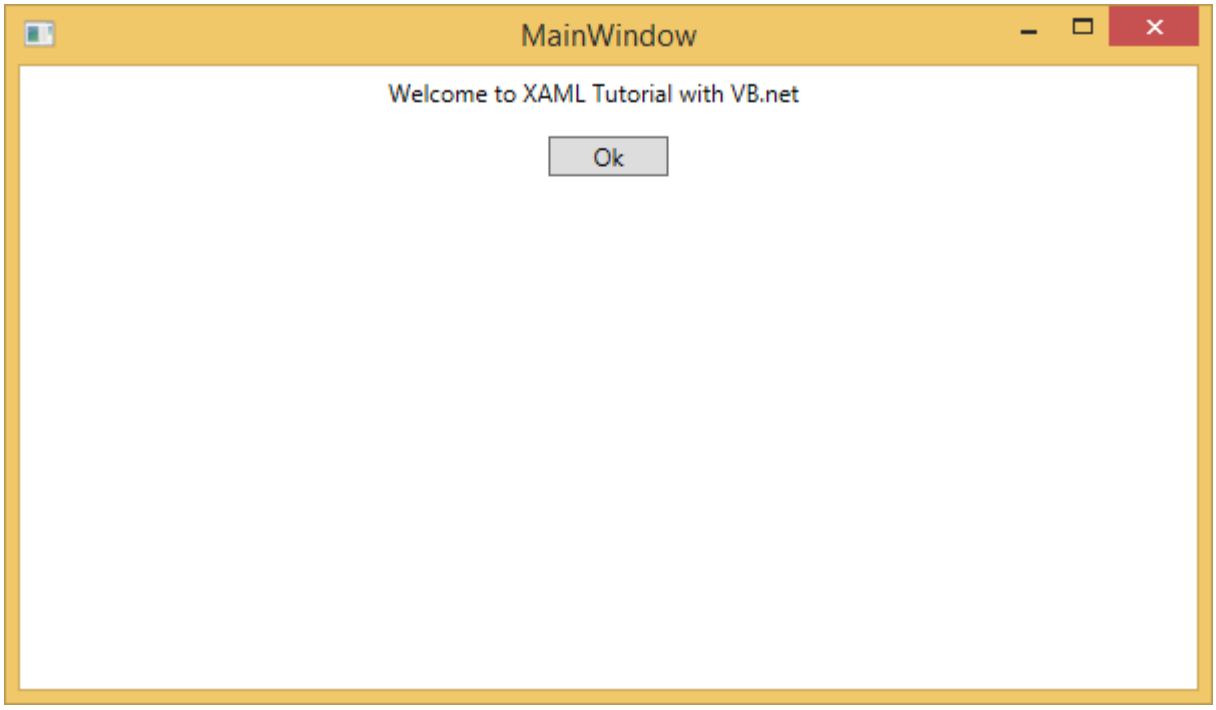
5. XAML VS. VB.NET

In this chapter, we will write the same example in VB.Net so that those who are familiar with VB.Net can also understand the advantages of XAML.

Let's take a look at the the same example again which is written in XAML:

```
<Window x:Class="XAMLVsCode.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="604">

    <StackPanel>
        <TextBlock Text="Welcome to XAML Tutorial with VB.net" Height="20"
Width="220"
                   Margin="5"/>
        <Button Content="Ok" Height="20" Width="60" Margin="5"/>
    </StackPanel>
</Window>
```

In this example, we have created a stack panel with a button and a Text block and defined some of the properties of the button and the text block such as Height, Width, and Margin. When the above code is compiled and executed, it will produce the following output:

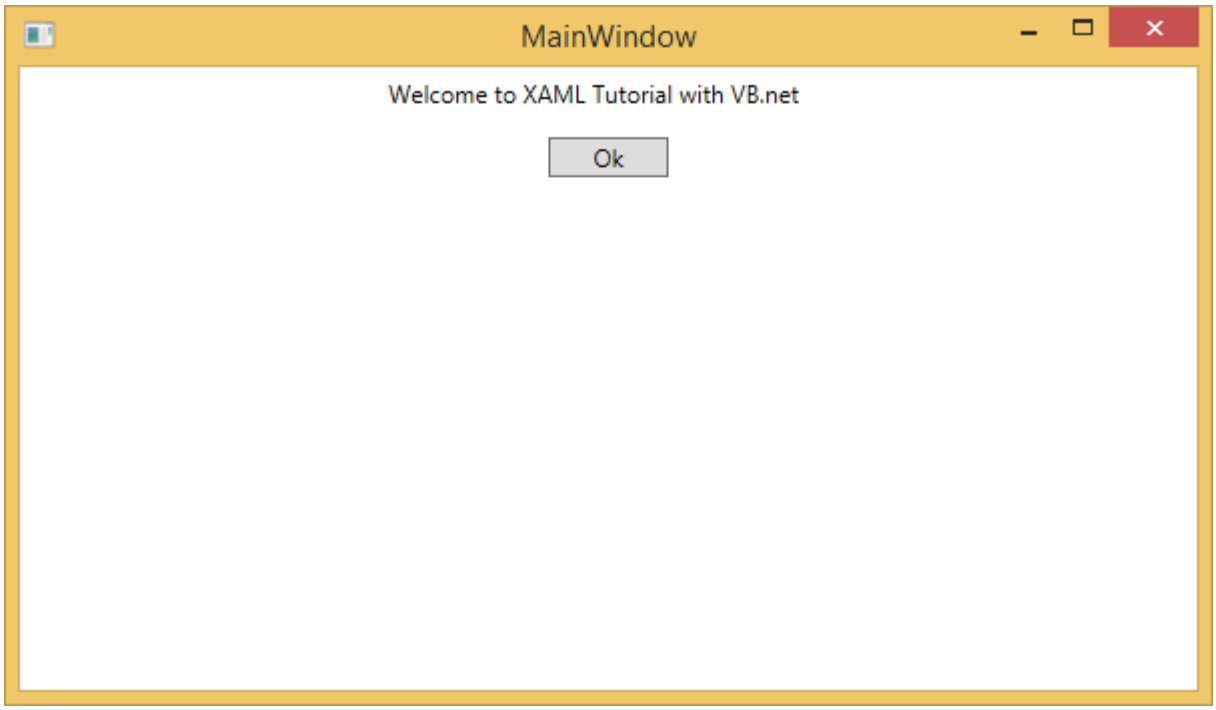


Now look at the same code which is written in VB.Net:

```
Public Class MainWindow

    Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Dim panel As New StackPanel()
        panel.Orientation = Orientation.Vertical
        Me.Content = panel
        Dim txtInput As New TextBlock
        txtInput.Text = "Welcome to XAML Tutorial with VB.net"
        txtInput.Width = 220
        txtInput.Height = 20
        txtInput.Margin = New Thickness(5)
        panel.Children.Add(txtInput)

        Dim btn As New Button()
        btn.Content = "Ok"
        btn.Width = 60
        btn.Height = 20
        btn.Margin = New Thickness(5)
```

```
        panel.Children.Add(btn)


    End Sub
End Class
```

When the above code is compiled and executed the output is exactly the same as the output of XAML code.



You can now visualize how simple it is to work with XAML as compared to VB.Net.

In the above example, we have seen that what we can do in XAML can also be done in other procedural languages such as C# and VB.Net.

Let's have a look at another example in which we will use both XAML and VB.Net. We will design a GUI in XAML and the behavior will be implemented in VB.Net.

In this example, a button is added to the main window. When the user clicks this button, it displays a message on the message box. Here is the code in XAML in which a Button Object is declared with some properties.

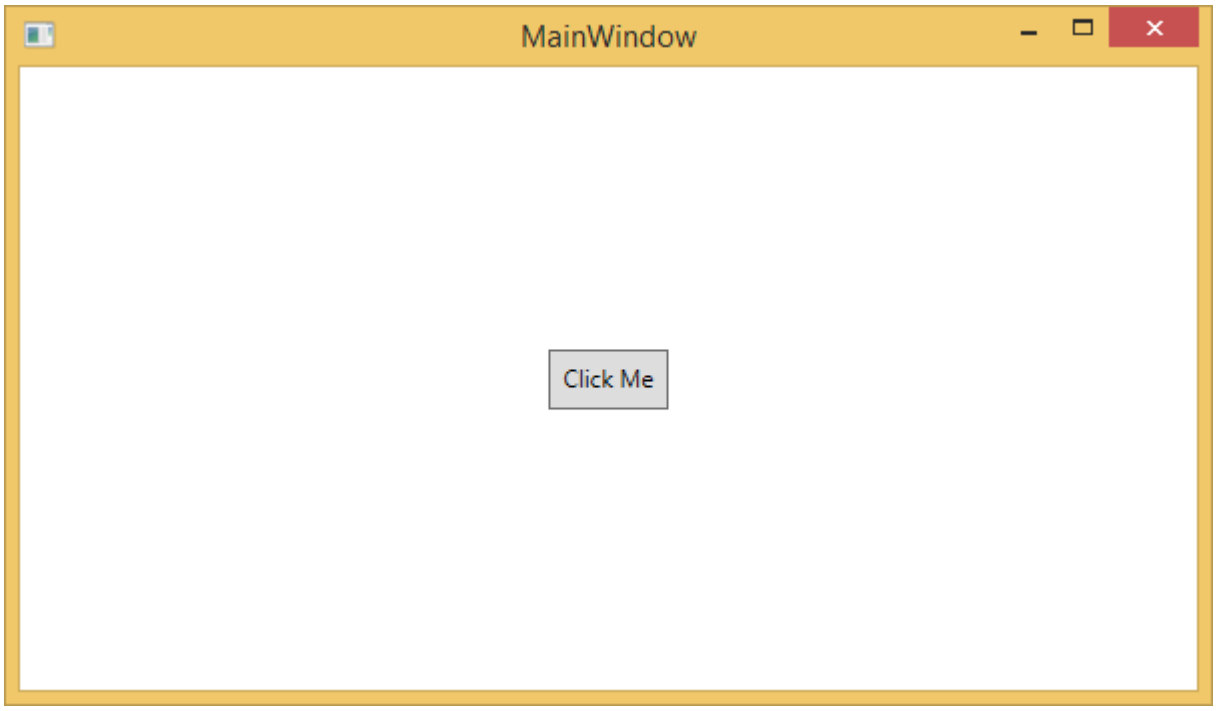
```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="604">
    <Grid>
        <Button Name="btn" HorizontalAlignment="Center" Width="60" Height="30"
Content="Click Me" />
```

```
    </Grid>
</Window>
```

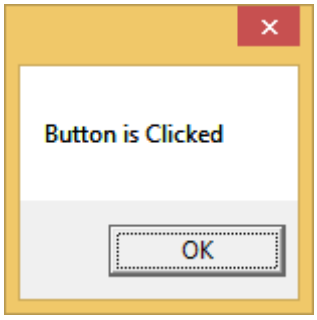
In VB.Net, the button click event (behavior) is implemented. This event displays the message on the messagebox.

```
Public Class MainWindow
    Private Sub btn_Click(sender As Object, e As RoutedEventArgs) Handles btn.Click
        MessageBox.Show("Button is Clicked")
    End Sub
End Class
```

When the above code is compiled and executed, it will display the following screen:



Now click on the above button that says "Click Me". It will display the following message:



6. XAML – BUILDING BLOCKS

This chapter will describe some of the basic and important building blocks of XAML applications. It will explain how

- to create and initialize an object,
- an object can be modified easily by using resources, styles, and templates,
- to make an object interactive by using transformations and animations.

Objects

XAML is a typically declarative language which can create and instantiate objects. It is another way to describe objects based on XML, i.e., which objects need to be created and how they should be initialized before the execution of a program. Objects can be

- Containers (Stack Panel, Dock Panel)
- UI Elements / Controls (Button, TextBox, etc.)
- Resource Dictionaries

Resources

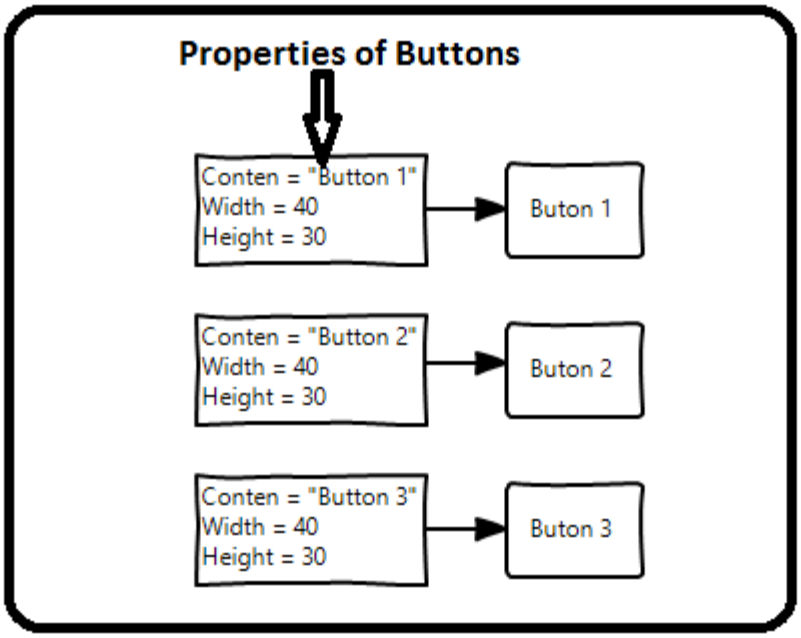
Resources are normally definitions connected with some object that you just anticipate to use more often than once. It is the ability to store data locally for controls or for the current window or globally for the entire applications.

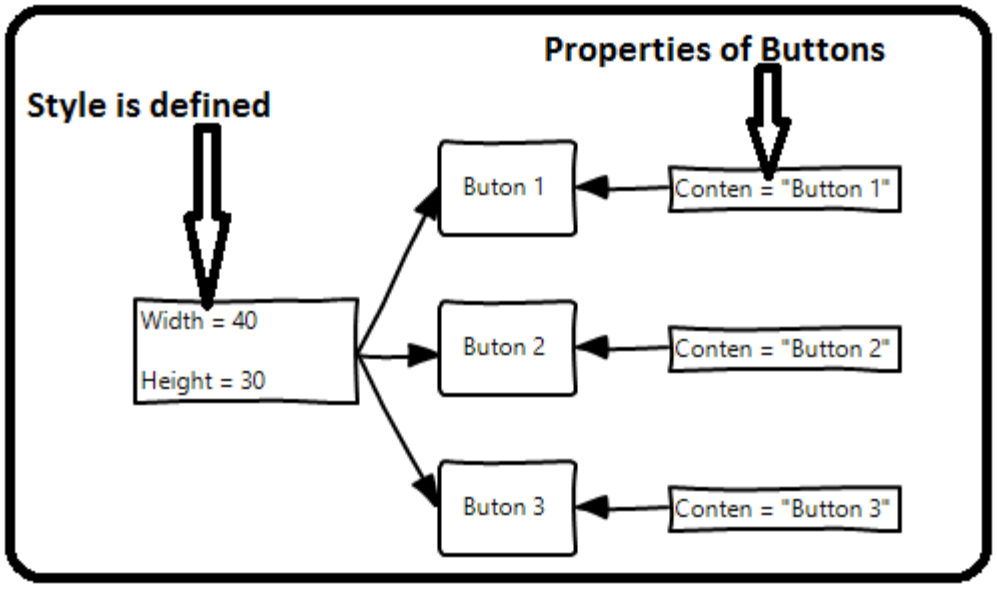
Styles

XAML framework provides several strategies to personalize and customize the appearance of an application. Styles give us the flexibility to set some properties of an object and reuse these specific settings across multiple objects for a consistent look.

- In styles, you can set only the existing properties of an object such as Height, Width, Font size, etc.
- Only the default behavior of a control can be specified.
- Multiple properties can be added into a style.

In the first diagram, you can see the same height and width properties are set for all the three button separately; but in the second diagram, you can see that height and width which are same for all the buttons are added to a style and then this style is associated with all the buttons.

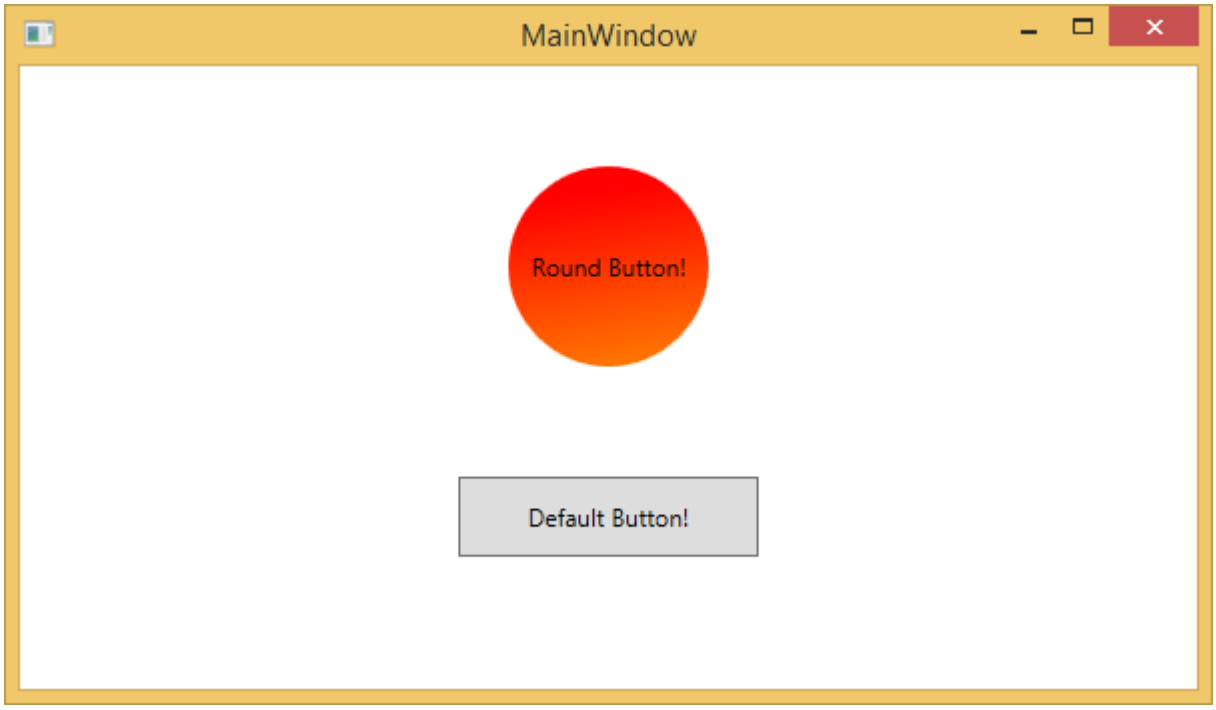




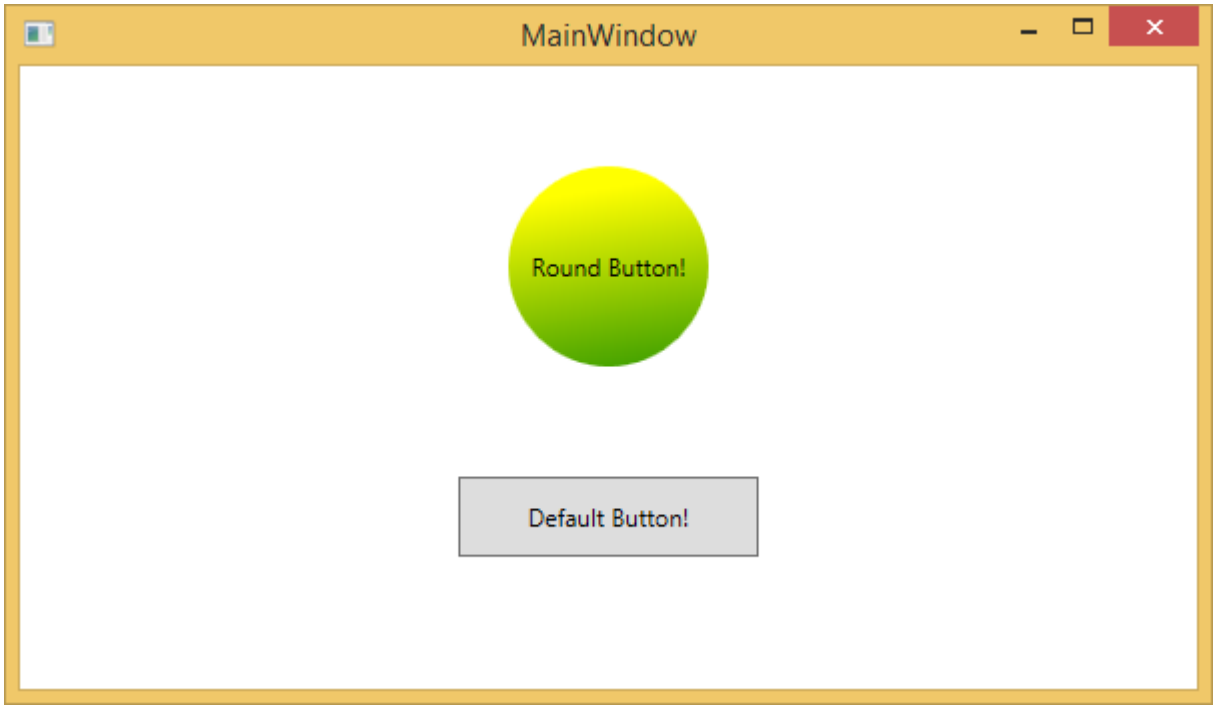
Templates

A template describes the overall look and visual appearance of a control. For each control, there is a default template associated with it which gives the appearance to that control. In XAML, you can easily create your own templates when you want to customize the visual behavior and visual appearance of a control.

In the following screenshot, there are two buttons, one is with template and the other one is the default button.



Now when you hover the mouse over the button, it also changes the color as shown below.



With templates, you can access more parts of a control than in styles. You can specify both existing and new behavior of a control.

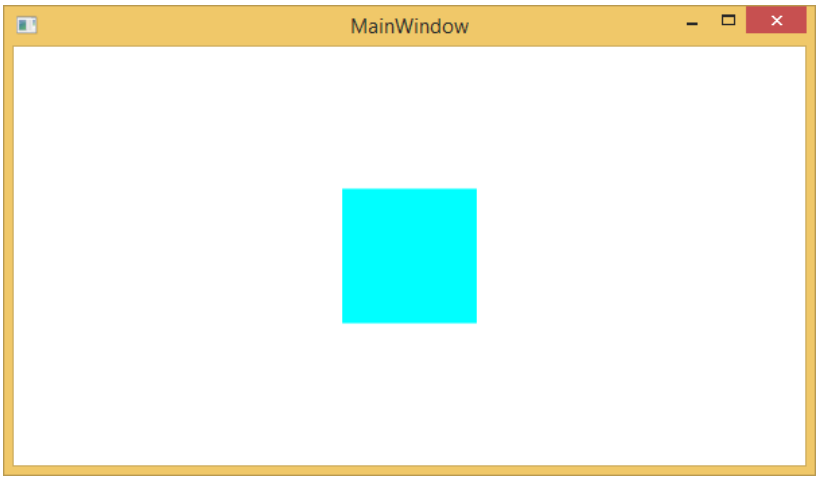
Animations and Transformations

Animations and transformations inside the Windows Runtime can improve your XAML application by building interactivity and movement. You can easily integrate the interactive

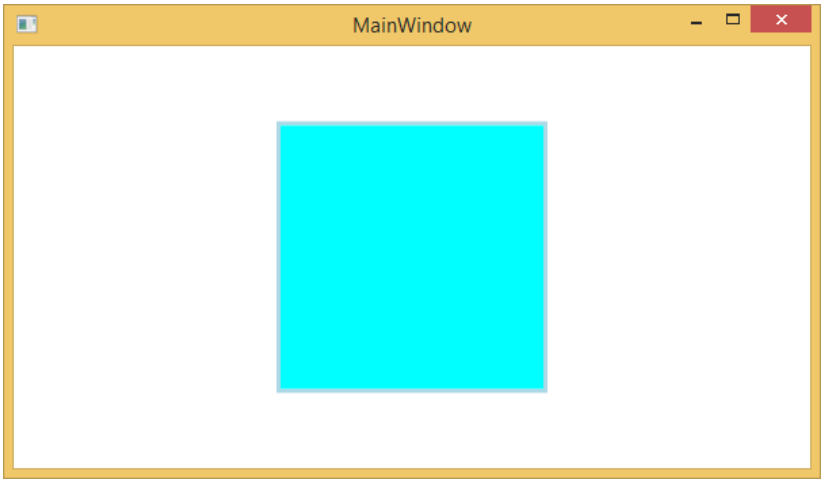
look and feel in your XAML application by using the animations from Windows Runtime animation library. Animations are used

- to enhance the user interface or to make it more attractive.
- to attract the attention of the user to a change.

In the following screenshot, you can see a square:



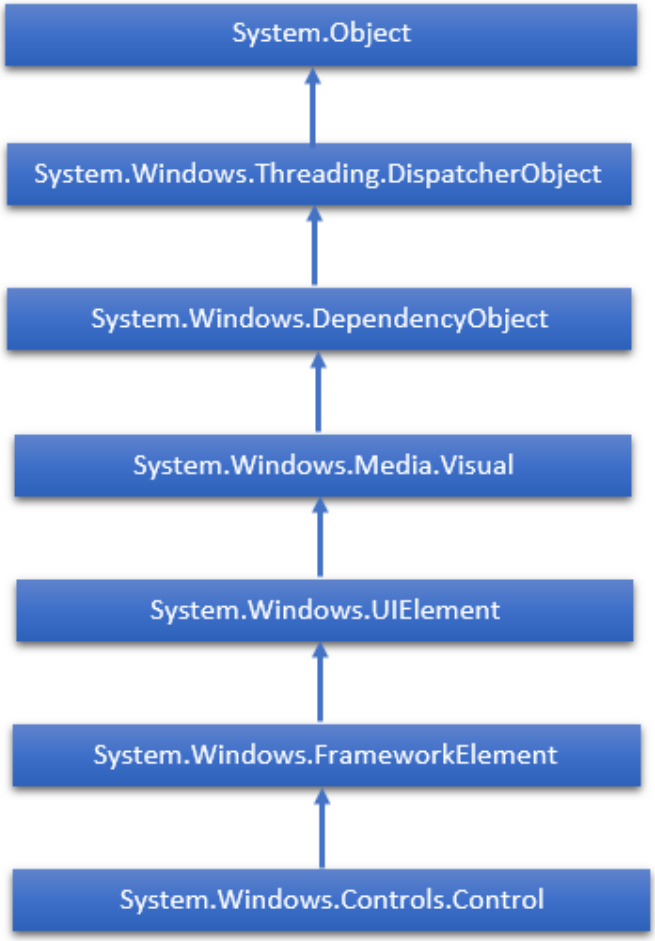
When you hover the mouse over this square, it will expend in all directions as shown below.



7. XAML – CONTROLS

The XAML User Interface framework offers an extensive library of controls that supports UI development for Windows. Some of them have a visual representation such Button, Textbox, TextBlock, etc.; while other controls are used as containers for other controls or content, for example, images. All the XAML controls are inherited from **System.Windows.Controls.Control**.

The complete inheritance hierarchy of controls is as follows:



Here is the list of controls which we will discuss one by one in this chapter.

Sr. No.	Controls & Description
1	**Button** A control that responds to user input.
2	**Calendar** Represents a control that enables a user to select a date by using a visual calendar display.
3	**CheckBox**

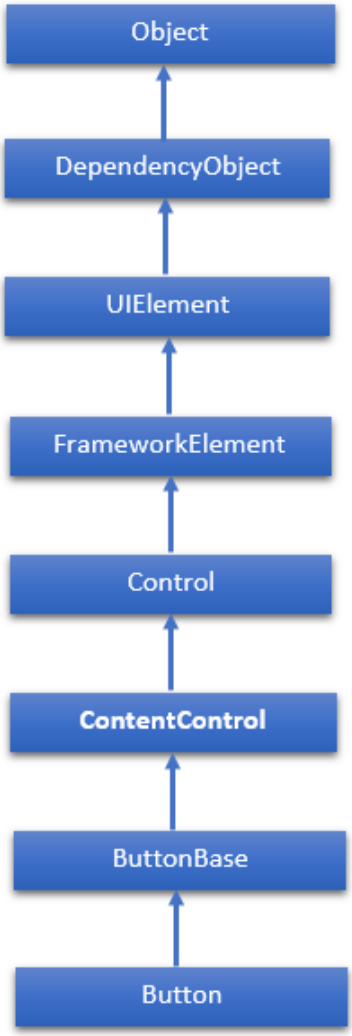
	A control that a user can select or clear.
4	**ComboBox** A drop-down list of items a user can select from.
5	**ContextMenu** Gets or sets the context menu element that should appear whenever the context menu is requested through a user interface (UI) from within this element.
6	**DataGrid** Represents a control that displays data in a customizable grid.
7	**DatePicker** A control that lets a user select a date.
8	**Dialogs** An application may also display additional windows to the user to gather or display important information.
9	**GridView** A control that presents a collection of items in rows and columns that can scroll horizontally.
10	**Image** A control that presents an image.
11	**ListBox** A control that presents an inline list of items that the user can select from.
12	**Menus** Represents a Windows menu control that enables you to hierarchically organize elements associated with commands and event handlers.
13	**PasswordBox** A control for entering passwords.
14	**Popup** Displays content on top of existing content, within the bounds of the application window.
15	**ProgressBar** A control that indicates progress by displaying a bar.
16	**ProgressRing** A control that indicates indeterminate progress by displaying a ring.
17	**RadioButton** A control that allows a user to select a single option from a group of options.
18	**RichEditBox** A control that lets a user edit rich text documents with content like formatted text, hyperlinks, and images.
19	**ScrollViewer** A container control that lets the user pan and zoom its content.
20	**SearchBox** A control that lets a user enter search queries.
21	**Slider** A control that lets the user select from a range of values by moving a Thumb control along a track.
22	**TextBlock** A control that displays text.
23	**TimePicker** A control that lets a user set a time value.
24	**ToggleButton** A button that can be toggled between 2 states.
25	**ToolTip**

	A pop-up window that displays information for an element.
26	**Window** The root window which provides minimize/maximize option, Title bar, border and close button.

In this chapter we will discuss all these controls with implementation.

Button

The Button class represents the most basic type of button control. The hierarchical inheritance of Button class is as follows:



Given below are the most commonly used properties of Button.

Sr. No.	Property & Description
1	**Background**

	Gets or sets a brush that provides the background of the control. (Inherited from Control)
2	**BorderBrush** Gets or sets a brush that describes the border fill of a control. (Inherited from Control)
3	**BorderThickness** Gets or sets the border thickness of a control. (Inherited from Control)
4	**Content** Gets or sets the content of a ContentControl. (Inherited from ContentControl)
5	**ClickMode** Gets or sets a value that indicates when the Click event occurs, in terms of device behavior. (Inherited from ButtonBase)
6	**ContentTemplate** Gets or sets the data template that is used to display the content of the ContentControl. (Inherited from ContentControl)
7	**FontFamily** Gets or sets the font used to display text in the control. (Inherited from Control)
8	**FontSize** Gets or sets the size of the text in this control. (Inherited from Control)
9	**FontStyle** Gets or sets the style in which the text is rendered. (Inherited from Control)
10	**FontWeight** Gets or sets the thickness of the specified font. (Inherited from Control)
11	**Foreground** Gets or sets a brush that describes the foreground color. (Inherited from Control)
12	**Height** Gets or sets the suggested height of a FrameworkElement. (Inherited from FrameworkElement)
13	**HorizontalAlignment** Gets or sets the horizontal alignment characteristics that are applied to a FrameworkElement when it is composed in a layout parent, such as a panel or items control. (Inherited from FrameworkElement)
14	**IsEnabled** Gets or sets a value indicating whether the user can interact with the control. (Inherited from Control)
15	**IsPressed** Gets a value that indicates whether a ButtonBase is currently in a pressed state. (Inherited from ButtonBase)
16	**Margin** Gets or sets the outer margin of a FrameworkElement. (Inherited from FrameworkElement)
17	**Name** Gets or sets the identifying name of the object. When a XAML processor creates the object tree from XAML markup, run-time code can refer to the XAML-declared object by this name. (Inherited from FrameworkElement)
18	**Opacity** Gets or sets the degree of the object's opacity. (Inherited from UIElement)
19	**Resources** Gets the locally defined resource dictionary. In XAML, you can establish resource items as child object elements of a frameworkElement. Resources

	property element, through XAML implicit collection syntax. (Inherited from FrameworkElement)
20	**Style** Gets or sets an instance Style that is applied for this object during layout and rendering. (Inherited from FrameworkElement)
21	**Template** Gets or sets a control template. The control template defines the visual appearance of a control in UI, and is defined in XAML markup. (Inherited from Control)
22	**VerticalAlignment** Gets or sets the vertical alignment characteristics that are applied to a FrameworkElement when it is composed in a parent object such as a panel or items control. (Inherited from FrameworkElement)
23	**Visibility** Gets or sets the visibility of a UIElement. A UIElement that is not visible is not rendered and does not communicate its desired size to layout. (Inherited from UIElement)
24	**Width** Gets or sets the width of a FrameworkElement. (Inherited from FrameworkElement)

End of ebook preview

If you liked what you saw…

Buy it from our store @ https://store.tutorialspoint.com