Windows 10

Apps Development

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Welcome to Windows 10 tutorial. This tutorial is designed for people who want to learn how to develop apps meant for Windows 10. After completing it, you will have a better understating of Windows apps and what you can do with Windows applications using XAML and C#.

Audience

This tutorial has been prepared for anyone who has a basic knowledge of XAML, C#, and Visual Studio and has an urge to develop apps for mobile or desktop.

Prerequisites

Before you start proceeding with this tutorial, we are assuming that you have a good understanding of the basics of XAML, C#, and Visual Studio. If you are not well aware of these concepts, then we will suggest you to go through our short tutorials on these topics.

Copyright & Disclaimer

Table of Contents

1. Introduction

This tutorial is designed for people who want to learn how to develop Windows 10 applications. In this tutorial, we are going to learn -

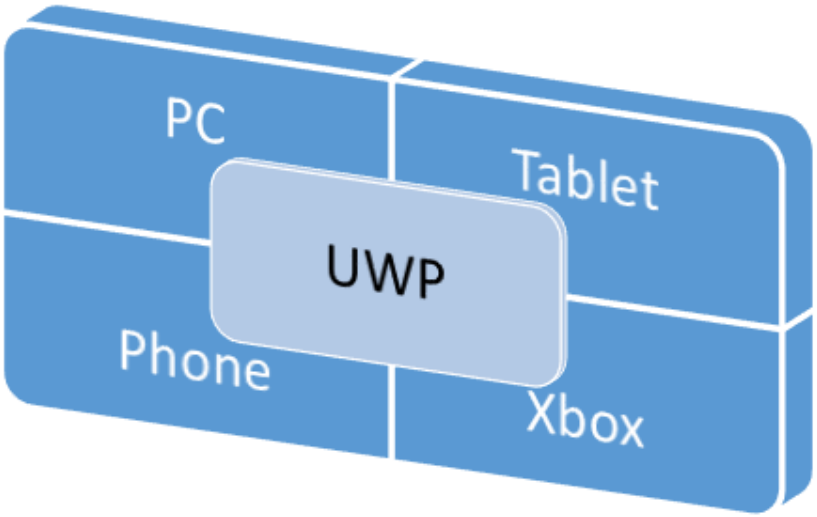
- Windows 10 application development
- Updates of the new OS released by Microsoft
- New features for the developers in the updates

A lot of interesting app scenarios are now possible that were not available to us in the first release. Microsoft has not only added new APIs, they have also extended the existing APIs.

Universal Windows app

A Universal Windows app was first introduced in Windows 8 as the Windows Runtime, which was built upon the Universal Application Platform.

Now, in Windows 10, the name of the Universal Application Platform has been changed to Universal Windows Platform (UWP). You can build modern and fully immersive apps by targeting Windows 10 devices for Windows Store such as PC, tablet, phone, etc.



In Windows 10, you can easily develop applications to reach all the devices supported on Windows 10 with just –

- One API set
- One app package
- And one store

The Universal Windows Platform also supports different screen sizes and different interaction models such as touch pad, mouse & keyboard, a game controller, or a pen.

Characteristics of UWP apps

Here are some of the characteristics of Universal Windows apps, which make it superior to Windows 10.

- You can target device families and not OS like Windows 8.1.

- Apps are packaged and distributed using the **.AppX** packaging format, which ensures that your apps can be deployed and updated seamlessly.

- You can submit your application to the Windows store and it will make it available on all device families, or only those devices you choose. You can easily manage all your apps for Windows devices in one place.

- You can limit the availability of your application to the particular device family.

- The core APIs of Universal Windows Platform (UWP) are the same across all Windows device families. So your app can run on all Windows 10 devices if it is uses only the core APIs.

- With the help of Extension SDKs, you can light up your application for particular devices.

Development Choices

Universal Windows applications can be created in any of the following languages:

- C# or Visual Basic with XAML
- JavaScript with HTML
- C++ with DirectX and/or XAML

You can also write components in one language and use them in an application that is developed in another language.

2. Windows 10 – UWP

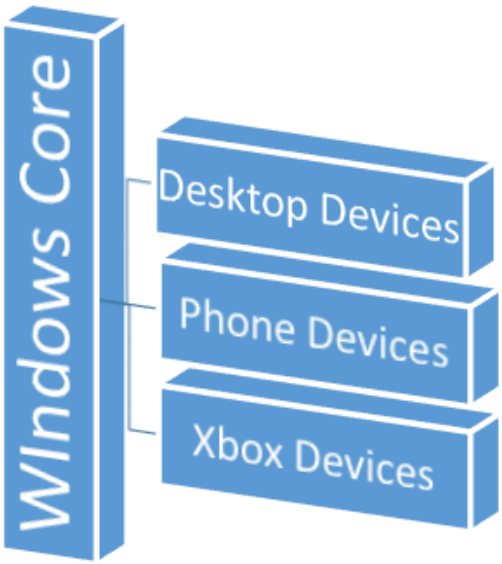
Windows Runtime (WinRT) is a platform-homogeneous application architecture, which supports development in C++/CX, C#, VB.NET and JavaScript. WinRT applications natively support both the x86 and ARM architectures. Some important features are.

- It was first introduced in Windows Server 2012 in September 2012.
- WinRT APIs provide access to all core platform features using JavaScript, C#, Visual Basic, and C++.
- WinRT components support multiple languages and APIs such as native, managed and scripting languages.

Universal Windows Platform (UWP)

A Universal Windows app is built upon Universal Windows Platform (UWP), which was first introduced in Windows 8 as the Windows Runtime. In Windows 10, Universal Windows Platform (UWP) was introduced, which further advances the Windows Runtime (WinRT) model.

- In Windows 8.1, WinRT, for the first time, was aligned between Windows Phone 8.1 applications and Windows 8.1 applications with the help of Universal Windows 8 apps to target both Windows phone and Windows application using a shared codebase.
-
- Windows 10 Unified Core, which is known as Windows Core now, has reached to a point where UWP, now, provides a common app platform available on every device that runs on Windows 10.

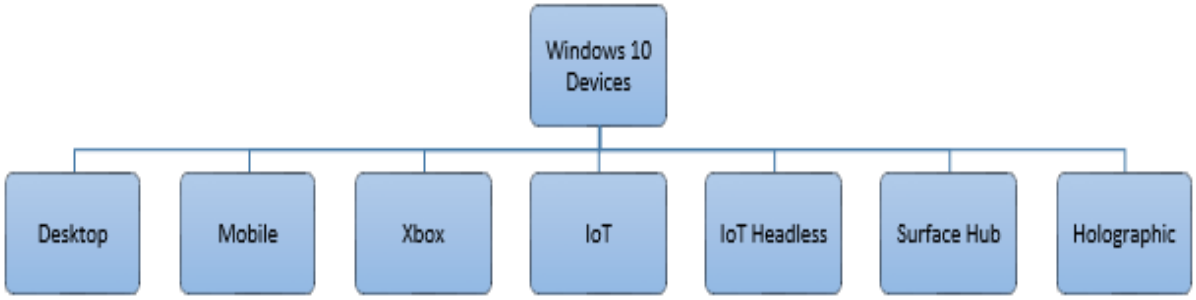


- UWP not only can call the WinRT APIs that are common to all devices, but also APIs (including Win32 and .NET APIs) that are specific to the device family that the app is running on.

Devices Supported by Windows 10

Windows 8.1 and Windows Phone 8.1 apps target an OS; either Windows or Windows Phone. Windows 10 applications do not target an OS but they target one or more device families.

Device families have their own APIs as well, which add functionality for that particular device family. You can easily determine all the devices, within a device family, on which your applications can be installed and run from the Windows Store. Here is the hierarchical representation of the device family.



Advantages of UWP

Universal Windows Platform (UWP) provides a handful of things for developers. They are:

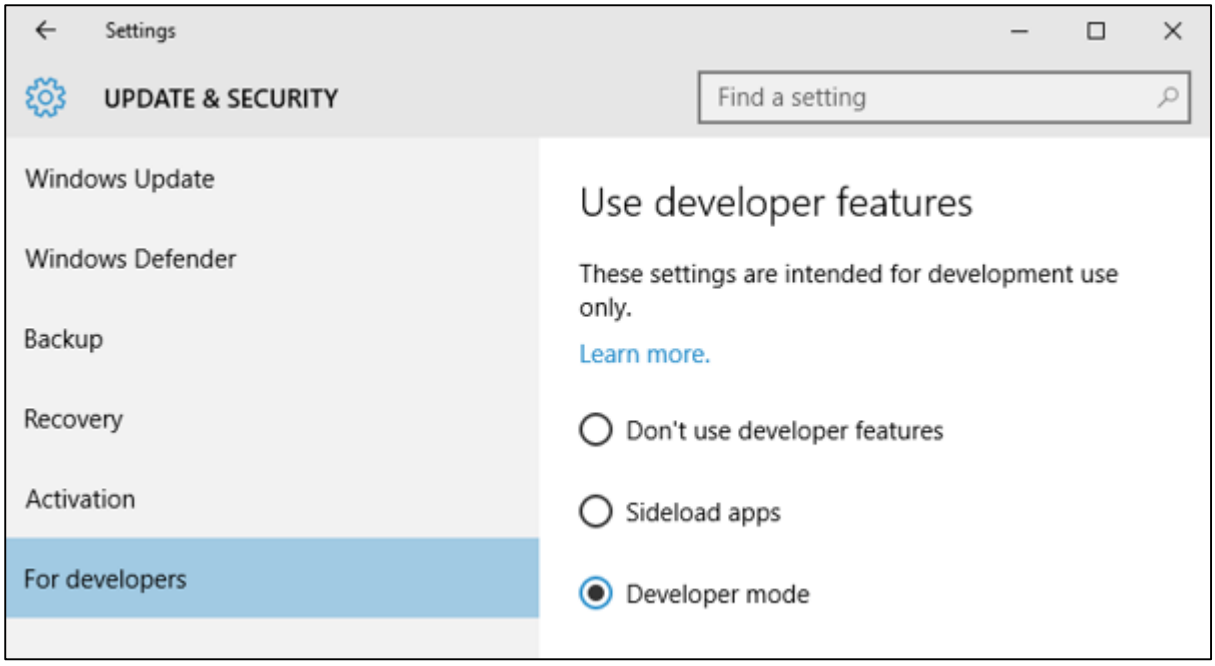
- One Operating System and One Unified Core for all the devices.
- One App Platform to run the applications across every family.
- One Dev Center to submit application and dashboard.
- One Store for all the devices.

Setup for UWP Development

The following steps need to be followed to start creating your own Universal Windows Platform (UWP) apps for Windows 10.

1. **Windows 10 OS:** UWP apps need the latest version of Windows to develop. You can also develop UWP applications on Windows 8.1 but there is no support for UI designer Window.
•
2. **Windows 10 developer tools:** In Visual studio 2015, you can design, code, test, and debug your UWP apps. You can download and install the free Microsoft Visual Studio Community 2015 from https://dev.windows.com/en-us/downloads
•
3. Enable development mode for Windows 10:
    - Go to Start > **Settings**.
    - Select **Update & security**.
    - Then select "**For developers"**.
    - Click on the **Developer mode**.

For UWP apps, it is important to test your applications on devices.



4. **Register as an app developer:** You can start developing apps, but to submit your apps to the store, you need a developer account. You can create your developer account here **https://msdn.microsoft.com/en-us/library/windows/apps/bg124287.aspx**

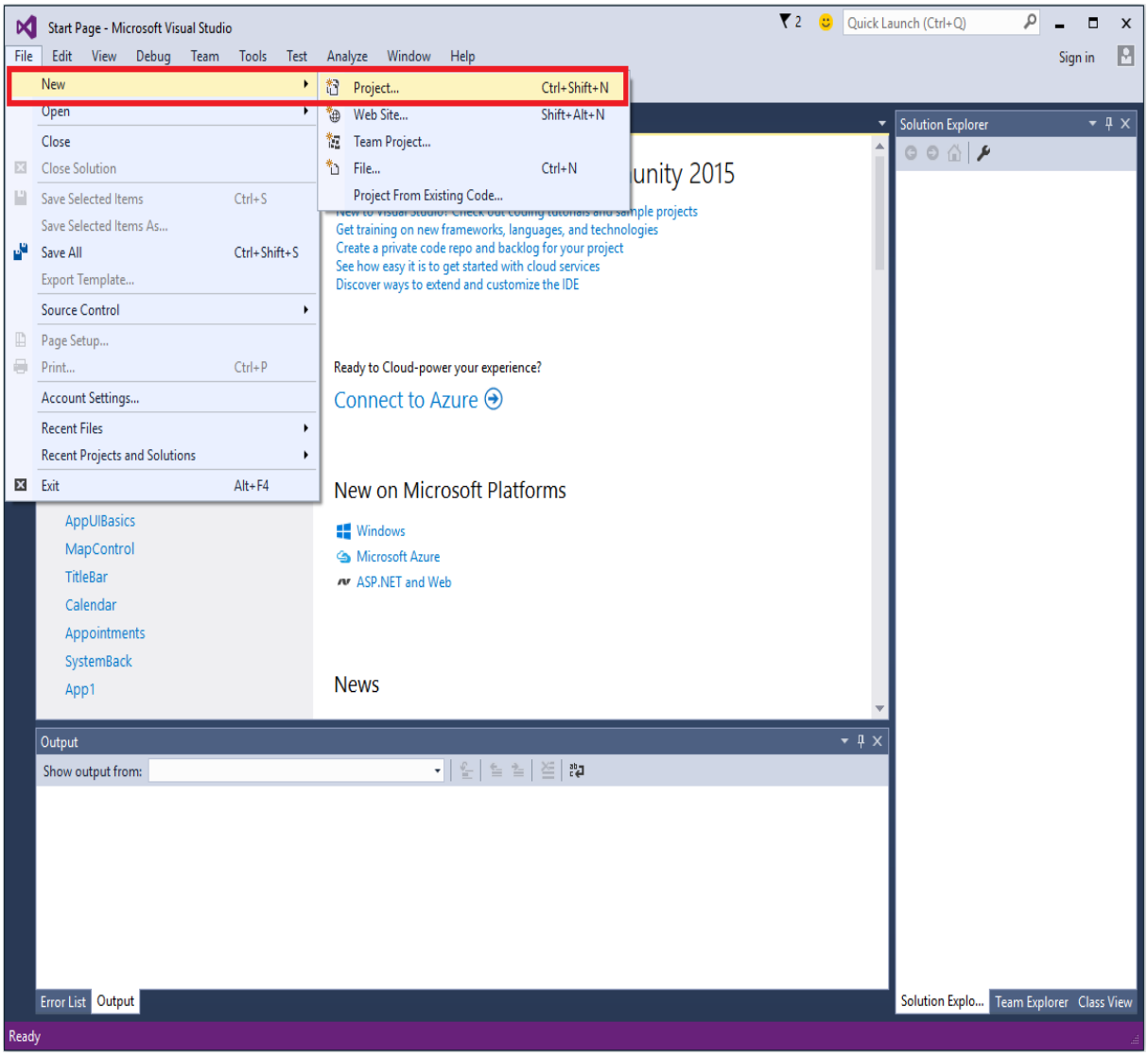
After following the above steps, you are now ready to start the development of a Universal Windows Platform (UWP) application.

3. Windows 10 – First App

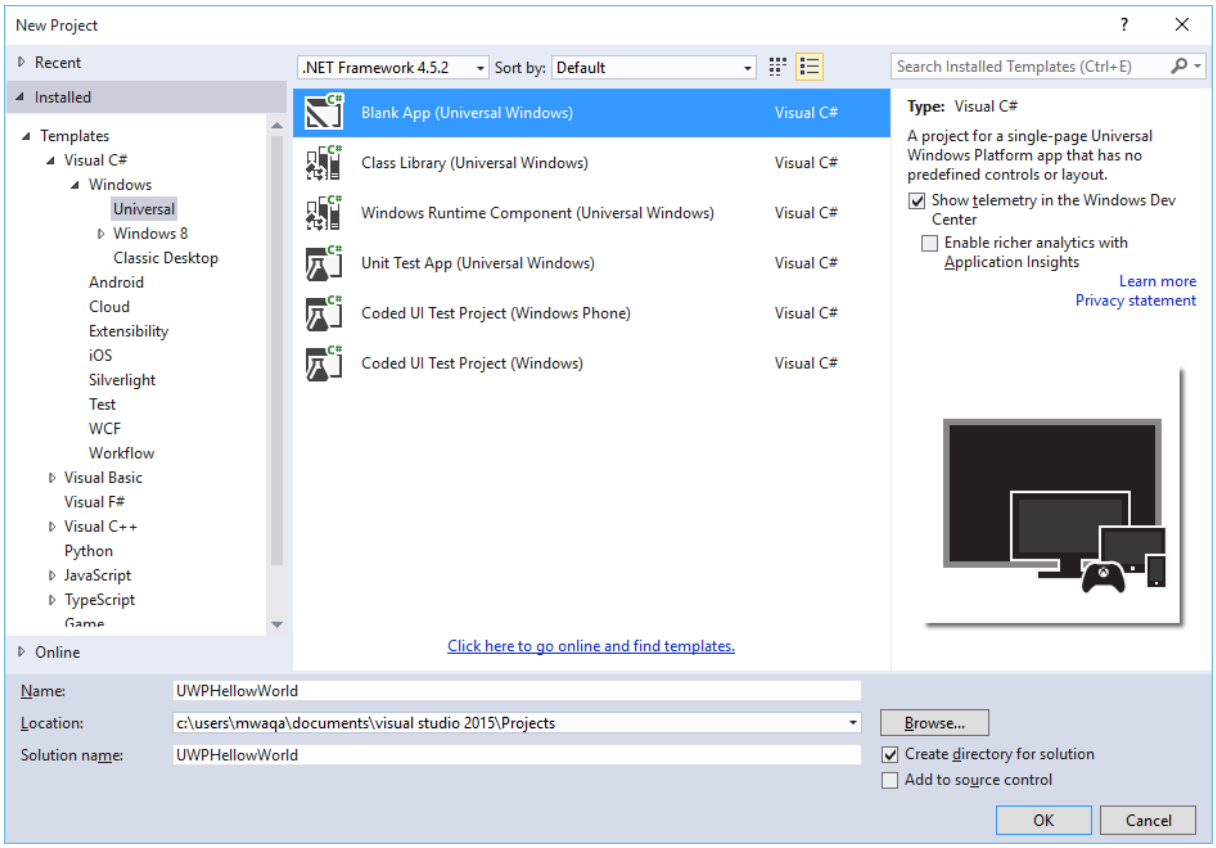
In this chapter, we will be creating our first simple application **"Hello world"** in Universal Windows Platform (UWP) using XAML and C# on Windows 10. We will demonstrate how a single UWP application created in Visual Studio can be run and executed on any Windows 10 device.

Let us start creating the App by following the steps given below.

1. Launch Visual Studio 2015.
2. Click on the **File** menu and select **New > Project**.



3. The following **New Project** dialog window will be displayed. You can see the different types of templates on the left pane of the dialog box.

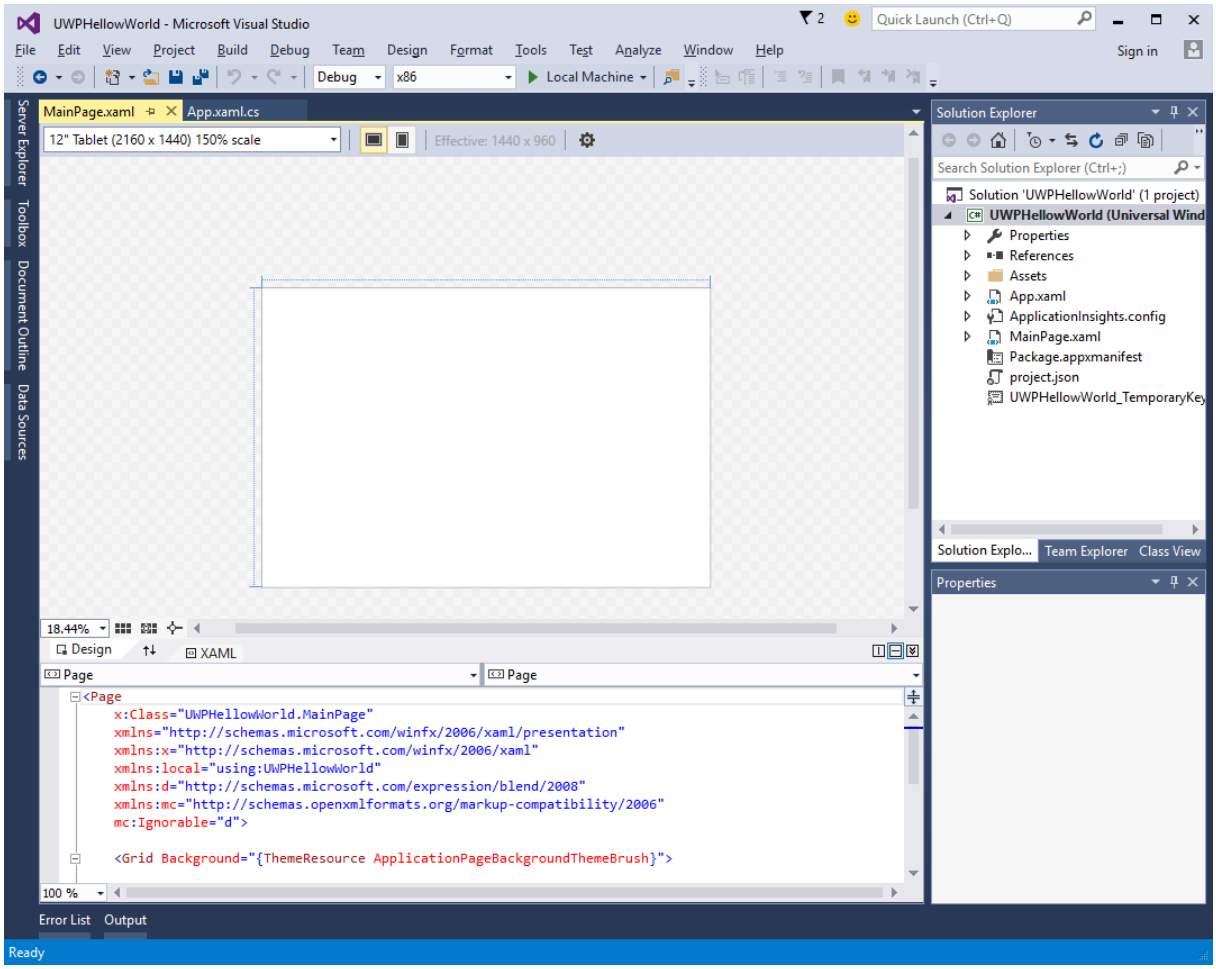


4. In the left pane, you can see the tree view. Select **Universal template** from **Templates > Visual C# > Windows**.

5. From the center pane, select the **Blank App (Universal Windows)** template.

6. Give a name to the project by writing **UWPHelloWorld** in the **Name field**.

7. Click **OK** to create a new UWP project.



8. You can see the newly created project in the **Solution Explorer.**

9. This is a blank app but it contains many files, which is the minimum requirement for any UWP application.

10. **MainPage.xaml** and **MainPage.xaml.cs** run when you execute your application.

11. By default, **MainPage.xaml** file contains the following information.

```
<Page
    x:Class="UWPHellowWorld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWPHellowWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
```

```
    <Grid Background=”{ThemeResource ApplicationPageBackgroundThemeBrush}”>


    </Grid>
</Page>
```

12. Given below is the default information available in **MainPage.xaml.cs.**

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;


// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409


namespace UWPHellowWorld
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }
    }
}
```

13. Let us add some Text Blocks, a textbox, and a button as shown in the XAML code below.

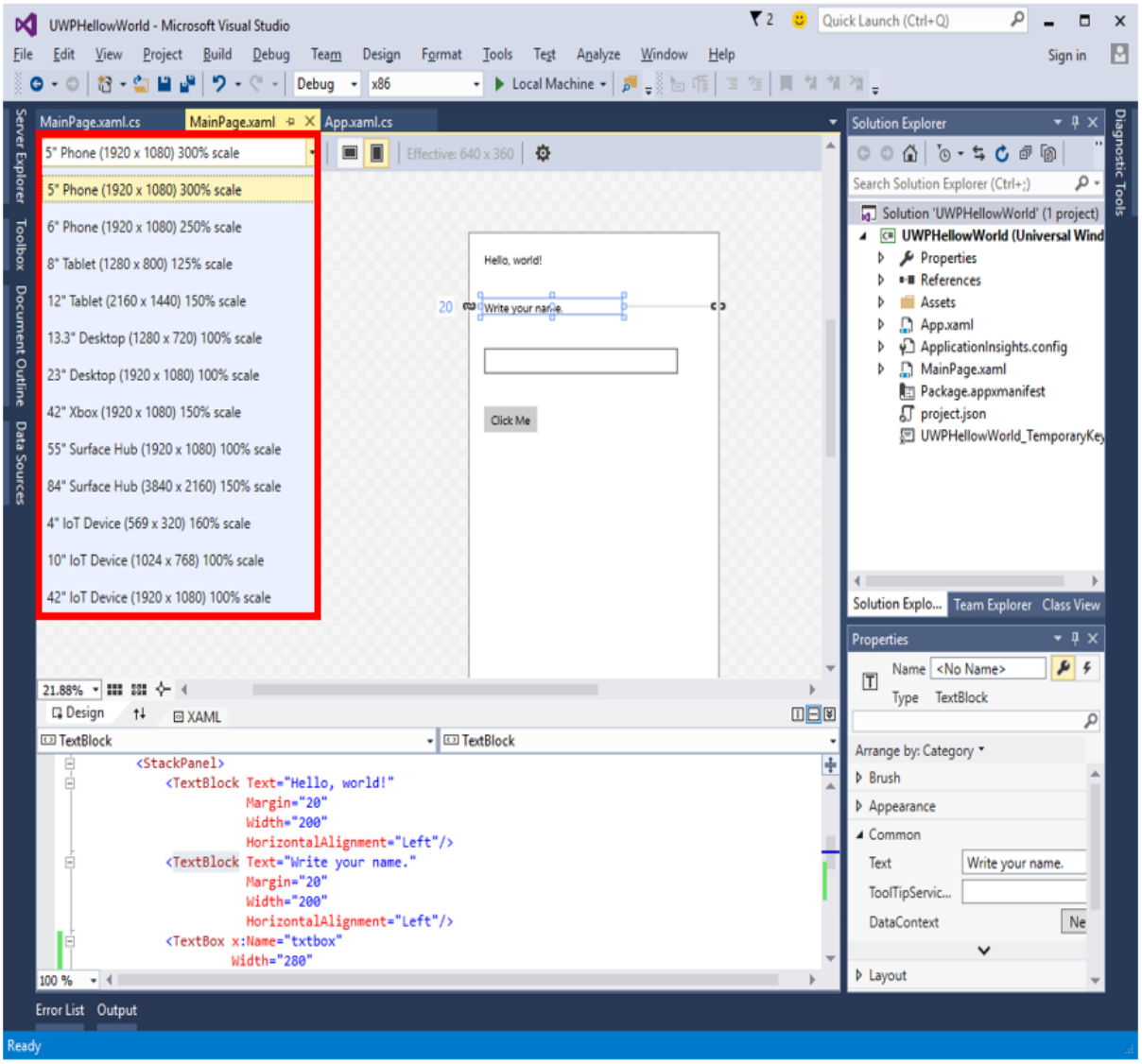
```
<Page
    x:Class="UWPHellowWorld.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWPHellowWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <StackPanel HorizontalAlignment="Center">
            <TextBlock Text="Hello, world!"
                       Margin="20"
                       Width="200"
                       HorizontalAlignment="Left"/>
            <TextBlock Text="Write your name."
                       Margin="20"
                       Width="200"
                       HorizontalAlignment="Left"/>
            <TextBox x:Name="txtbox"
                     Width="280"
                     Margin="20"
                     HorizontalAlignment="Left"/>
            <Button x:Name="button" Content="Click Me"
                    Margin="20"
                    Click="button_Click"/>
            <TextBlock x:Name="txtblock"
                       HorizontalAlignment="Left"
                       Margin="20"/>
        </StackPanel>
    </Grid>
</Page>
```

14. Given below is the click-event button in C#.

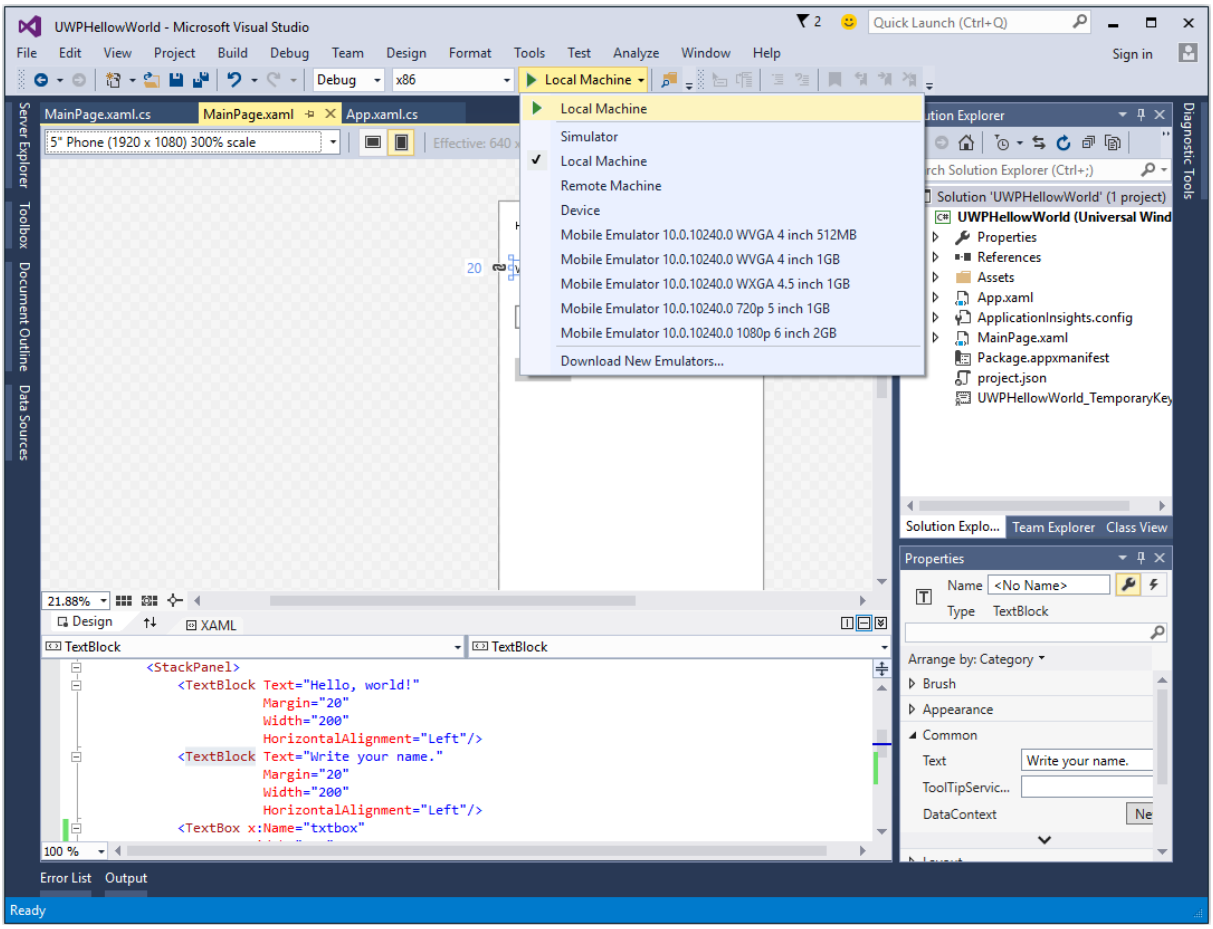
```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;


// The Blank Page item template is documented at
http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409


namespace UWPHellowWorld
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }


        private void button_Click(object sender, RoutedEventArgs e)
        {
            if (txtbox.Text != “”)
                txtblock.Text = “Hello: “ + txtbox.Text;
            else
```

```
                txtblock.Text = “You have not write your name”;
            }
        }
    }
```

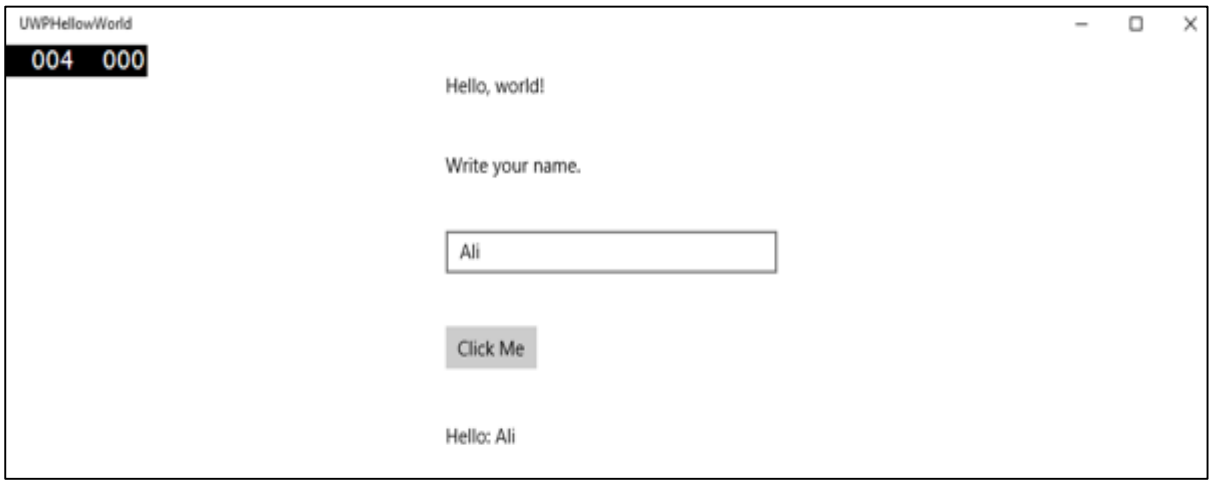
15. In the UWP project, **device preview** option is available on the **Design Window**, with the help of which you can change the layout easily, to fit into the screen size of all the devices in a device family you are targeting for your application.



16. You can run and test your app either on a local machine, a simulator or an emulator, or on a remote device. You can select the target device from the following menu as shown below:



17. Let us run the above code on a local machine and you will see the following window. Now, write any name in the text box and click the button **Click Me**.



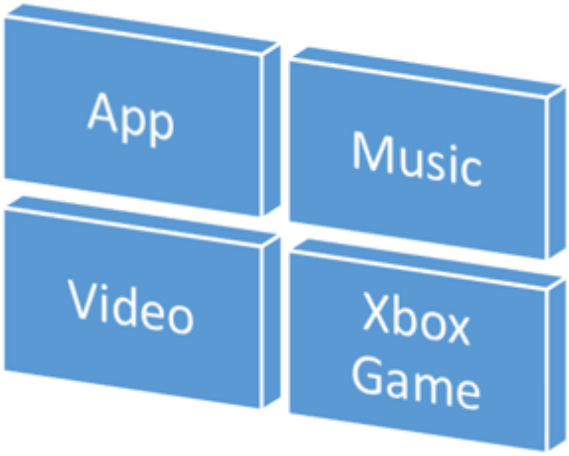
18. Now, if you want to test your app on an emulator, you can select a particular emulator from the menu and execute your application. You will see the following emulator:

We recommend you to execute the above application with different devices.

4. Windows 10 – Store

The benefit of Windows Store for developers is that you can sell your application. You can submit your single application for every device family.

- The Windows 10 Store is where applications are submitted, so that a user can find your application.
-
- In Windows 8, the Store was limited to application only and Microsoft provides many stores i.e. Xbox Music Store, Xbox Game Store etc.



- In Windows 8, all these were different stores but in Windows 10, it is called Windows Store. It is designed in a way where users can find a full range of apps, games, songs, movies, software and services in one place for all Windows 10 devices.

Monetization

Monetization means selling your app across desktop, mobile, tablets and other devices. There are various ways that you can sell your applications and services on Windows Store to earn some money.

You can select any of the following methods:

- The simplest way is to submit your app on store with paid download options.
-
- The Trails option, where users can try your application before buying it with limited functionality.
-
- Add advertisements to your apps with Microsoft Advertising.

Microsoft Advertising

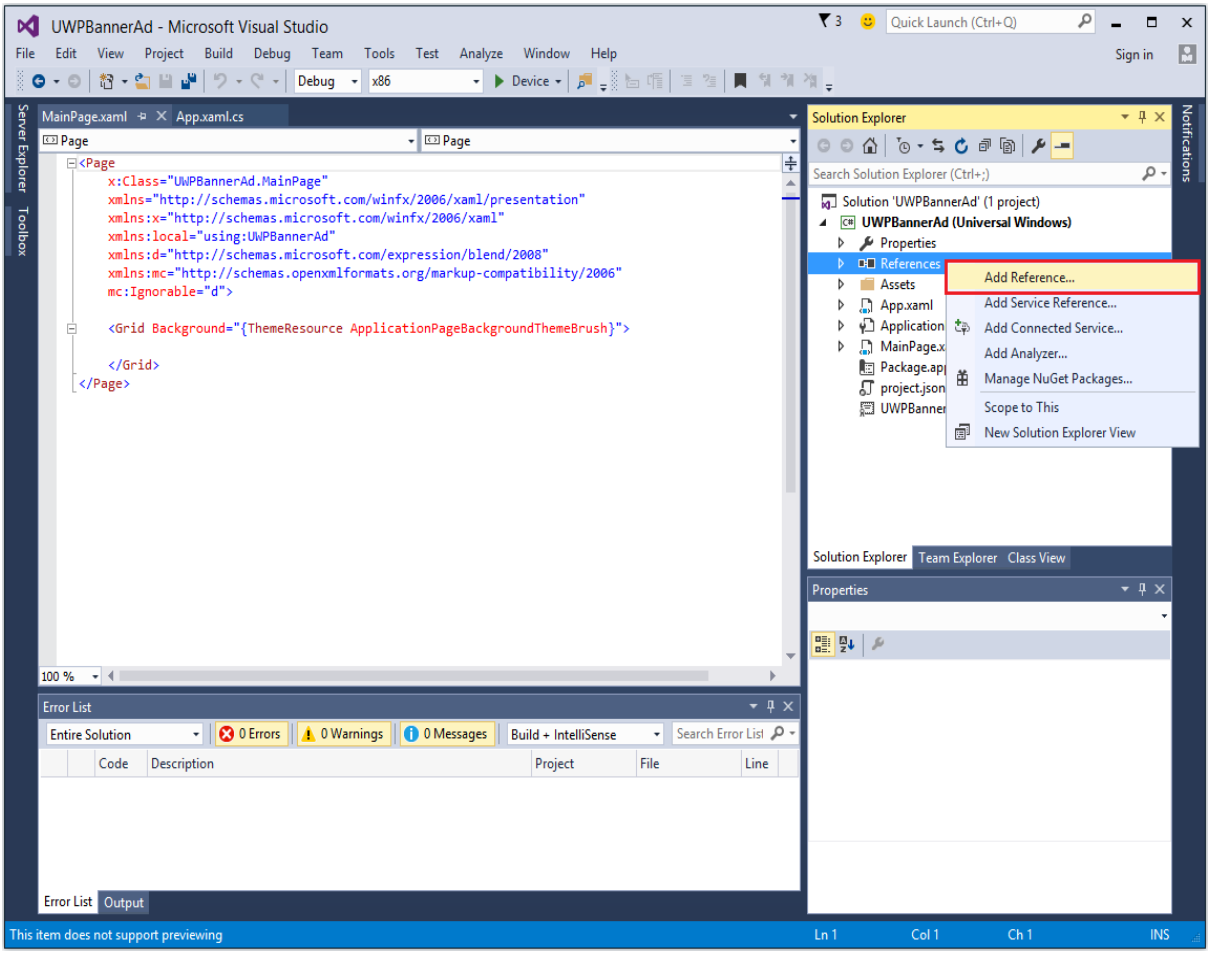
When you add Ads to your application and a user clicks on that particular Ad, then the advertiser will pay you the money. Microsoft Advertising allows developers to receive Ads from Microsoft Advertising Network.

- The Microsoft Advertising SDK for Universal Windows apps is included in the libraries installed by Visual Studio 2015.
-
- You can also install it from https://visualstudiogallery.msdn.microsoft.com/401703a0-263e-4949-8f0f-738305d6ef4b
-
- Now, you can easily integrate video and banner Ads into your apps.

Let us have a look at a simple example in XAML, to add a banner Ad in your application using **AdControl**.

1. Create a new Universal Windows blank app project with the name **UWPBannerAd.**

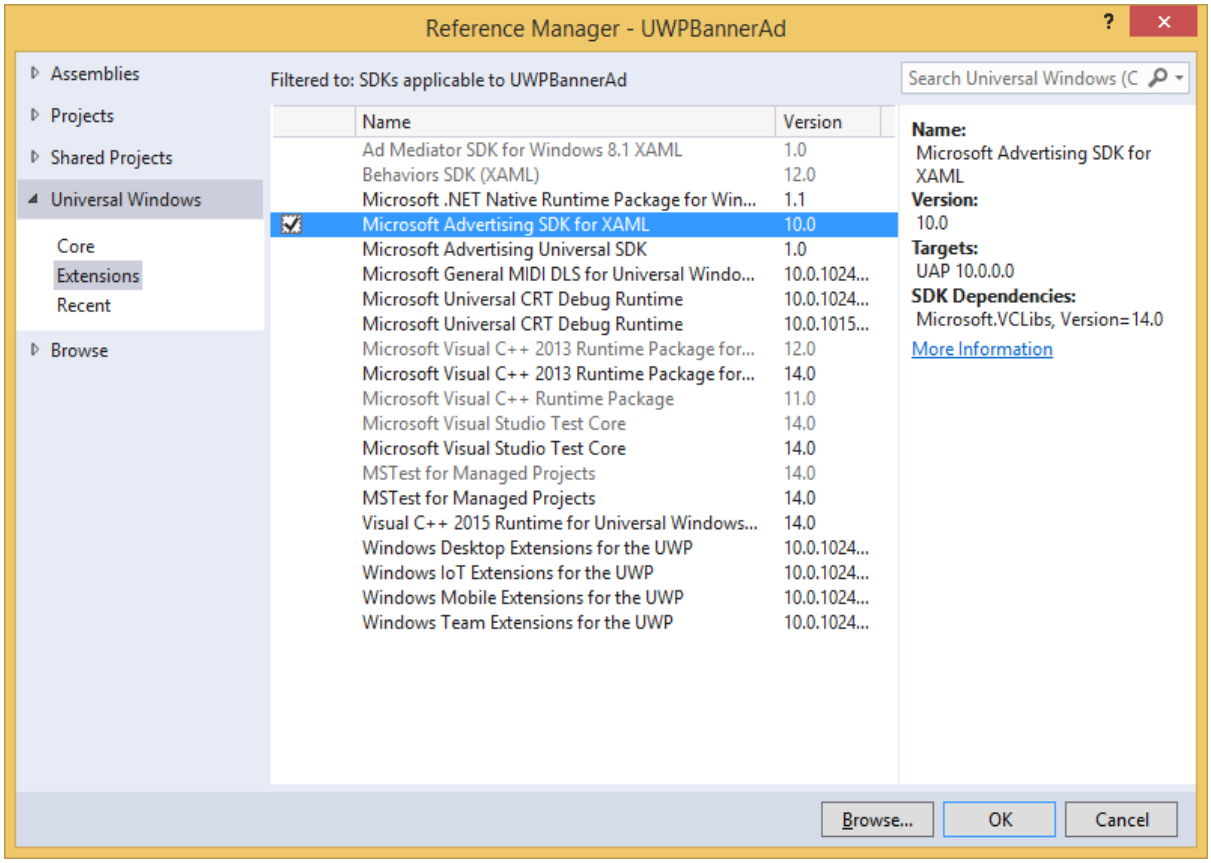
2. In the Solution Explorer, right click on References.



3. Select **Add References,** which will open the **Reference Manager** dialog.

•

4. From the left pane, select **Extensions** under Universal Windows option and check the **Microsoft Advertising SDK for XAML.**

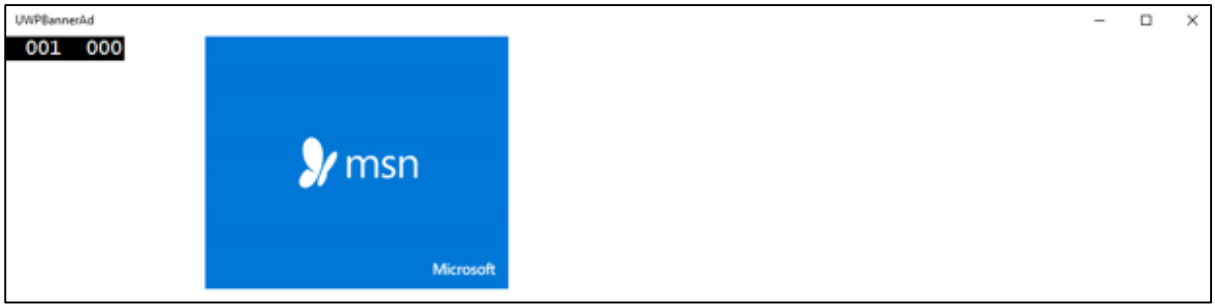


5. Click **OK** to Continue.
6. Given below is the XAML code in which **AdControl** is added with some properties.

```
<Page
    x:Class="UWPBannerAd.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWPBannerAd"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:UI="using:Microsoft.Advertising.WinRT.UI"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Center">
            <UI:AdControl ApplicationId="d25517cb-12d4-4699-8bdc-52040c712cab"
                          AdUnitId="10043121"
                          HorizontalAlignment="Left"
                          Height="580"
                          VerticalAlignment="Top"
```

```
                        Width="800"/>
        </StackPanel>
    </Grid>
</Page>
```

When the above code is compiled and executed on a local machine, you will see the following window with MSN banner on it. When you click this banner, it will open the MSN site.



You can also add a **video banner** in your application. Let us consider another example in which when the **Show ad** button is clicked, it will play the video advertisement of Xbox One.

Given below is the XAML code in which we demonstrate how a button is added with some properties and events.

```
<Page
    x:Class="UWPBannerAd.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWPBannerAd"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:UI="using:Microsoft.Advertising.WinRT.UI"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Center">
            <Button x:Name="showAd" Content="Show Ad"
                    HorizontalAlignment="Left"
                    Margin="138,296,0,0"
                    VerticalAlignment="Top"
                    FontSize="48"
                    Click="showAd_Click"/>
        </StackPanel>
```

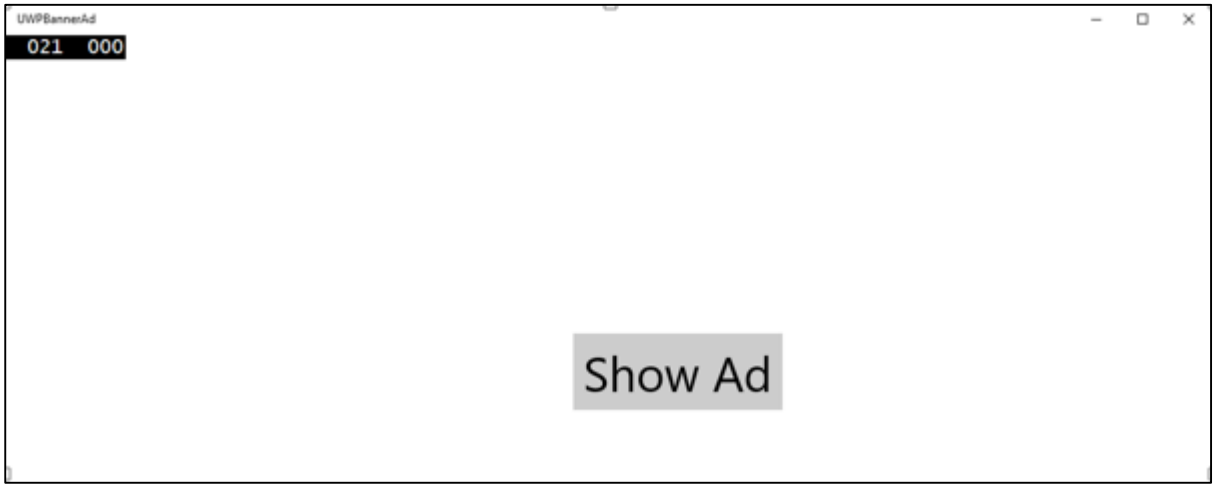
```
    </Grid>
</Page>
```

Given below is the click event implementation in C#.

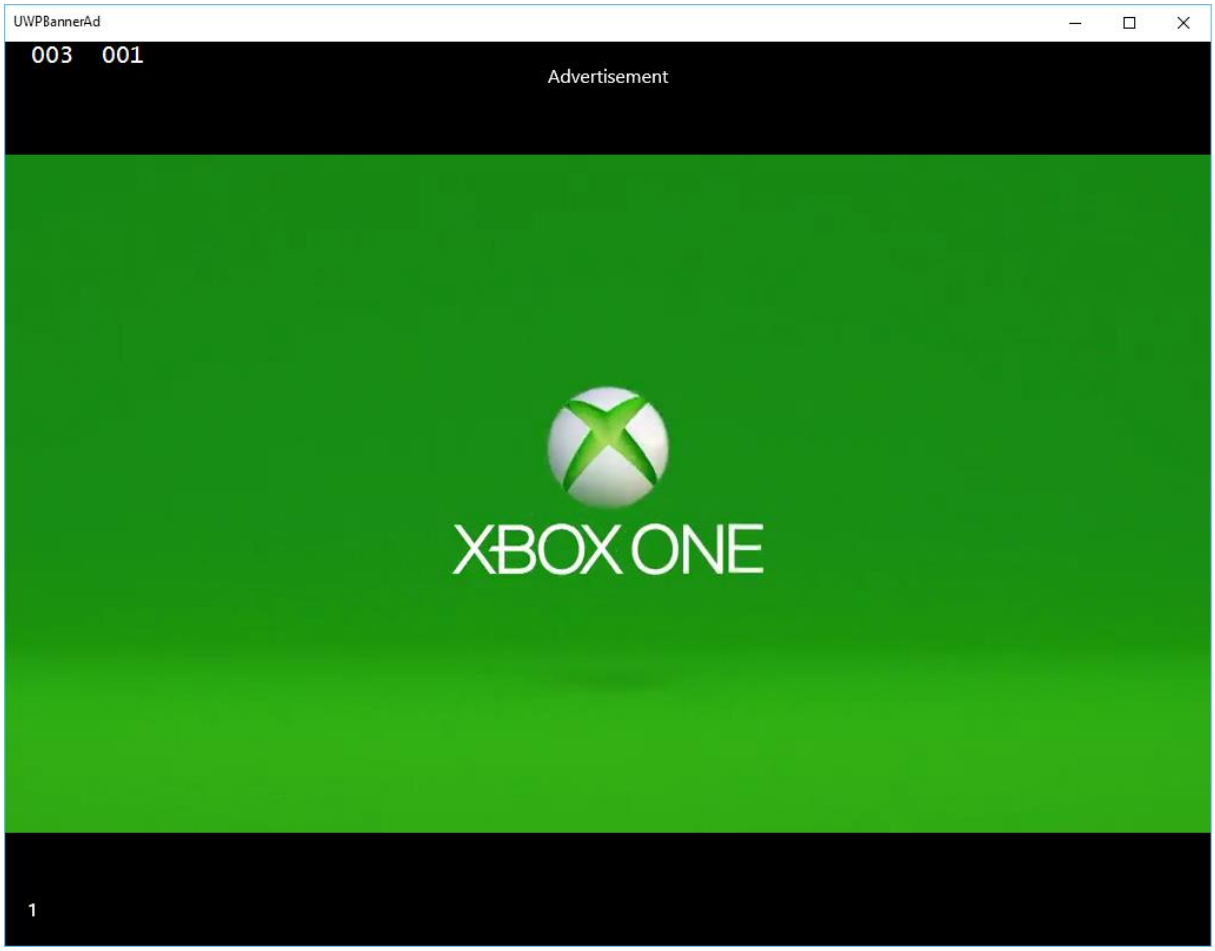
```
using Microsoft.Advertising.WinRT.UI;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

//     The     Blank     Page     item     template     is     documented     at
http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace UWPBannerAd
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        InterstitialAd videoAd = new InterstitialAd();
        public MainPage()
        {
            this.InitializeComponent();
        }

        private void showAd_Click(object sender, RoutedEventArgs e)
        {
            var MyAppId = "d25517cb-12d4-4699-8bdc-52040c712cab";
            var MyAdUnitId = "11388823";

            videoAd.AdReady += videoAd_AdReady;
            videoAd.RequestAd(AdType.Video, MyAppId, MyAdUnitId);
        }

        void videoAd_AdReady(object sender, object e)
        {
            if ((InterstitialAdState.Ready) == (videoAd.State))
            {
```

```
                videoAd.Show();
            }
        }
    }
}
```

When the above code is compiled and executed on a local machine, you will see the following window, which contains a **Show Ad** button.



Now, when you click on the **Show Ad** button, it will play the video on your app.



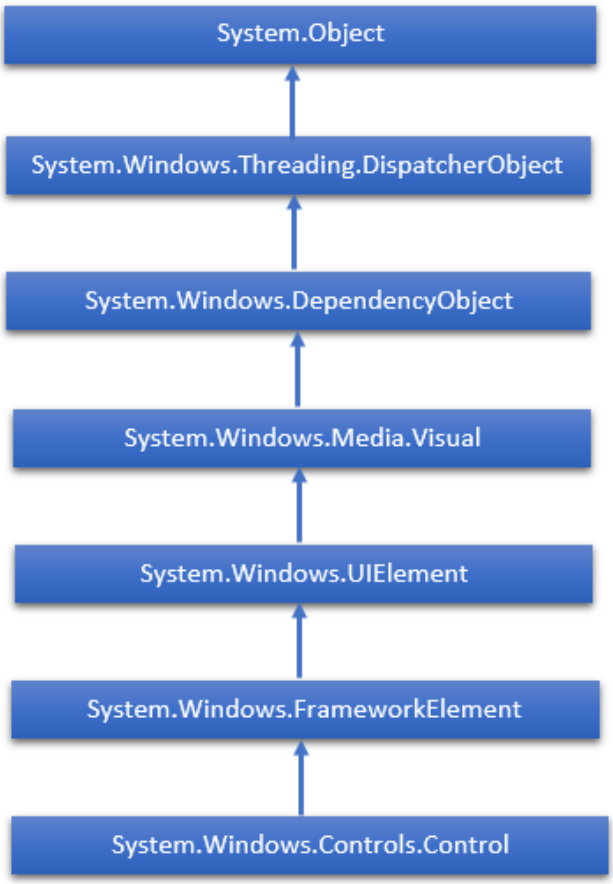
5. Windows 10 – XAML Controls

XAML Stands for Extensible Application Markup Language. It is a User Interface framework and it offers an extensive library of controls that support UI development for Windows. Some of them have a visual representation such as a Button, Textbox and TextBlock etc; while other controls are used as the containers for other controls or content, such as images etc. All the XAML controls are inherited from **"System.Windows.Controls.Control".**

XAML Emerging Story

XAML is used in many important Microsoft platforms such as the Windows Presentation Foundation (WPF), the Silverlight and now, Windows apps. Now, Microsoft Office 2016 is also a family of UWP apps. XAML is a rich Platform, which provides very cool features and controls that can be used in UWP applications.

The complete inheritance hierarchy of controls is shown below.



Layout Controls

Layout of Controls is very important and critical for application usability. It is used to arrange a group of GUI elements in your application. There are certain important things to consider while selecting the layout panels:

- Positions of the child elements.
- Sizes of the child elements.
- Layering of overlapping child elements on top of each other.

A list of **Layout Controls** is given below:

Controls	Description
StackPanel	**StackPanel** is a simple and useful layout panel in XAML. In stack panel, child elements can be arranged in a single line either horizontally or vertically based on orientation property.
WrapPanel	In **WrapPanel,** child elements are positioned in sequential order from left to right or from top to bottom based on the orientation property. The only difference between StackPanel and WrapPanel is that it does not stack all the child elements into a single line but it wraps the remaining elements to another line if there is no space left.
DockPanel	**DockPanel** defines an area to arrange child elements relative to each other, either horizontally or vertically. With DockPanel you can easily dock child elements to top, bottom, right, left and center with Dock property. With **LastChildFill** property, the last child element fill the remaining space regardless of any other dock value when set for that element.
Canvas	**Canvas** is the basic layout panel in which child elements can be positioned explicitly using coordinates that are relative to any side such as left, right, top and bottom. Typically Canvas is used for 2D graphic elements (such as Ellipse, Rectangle etc.) but not for UI elements because specifying absolute coordinates give trouble while resizing, localizing or scaling in an XAML application.
Grid	**Grid** provides a flexible area, which consists of rows and columns. In Grid, child elements can be arranged in a tabular form. Elements can be added to any specific row and column by using **Grid.Row** and **Grid.Column** properties.
SplitView	**SplitView** represents a container with two views; one view for the main content and another view that is typically used for navigation commands.
RelativePanel	**RelativePanel** defines an area within which you can position and align child objects in relation to each other or the parent panel.
ViewBox	**ViewBox** defines a content decorator that can stretch and scale a single child to fill the available space.
FlipView	**FlipView** represents an item’s control that displays one item at a time, and enables "flip" behavior for traversing its collection of items.

GridView	**GridView** is a control that presents a collection of items in rows and columns and can be scrolled horizontally.