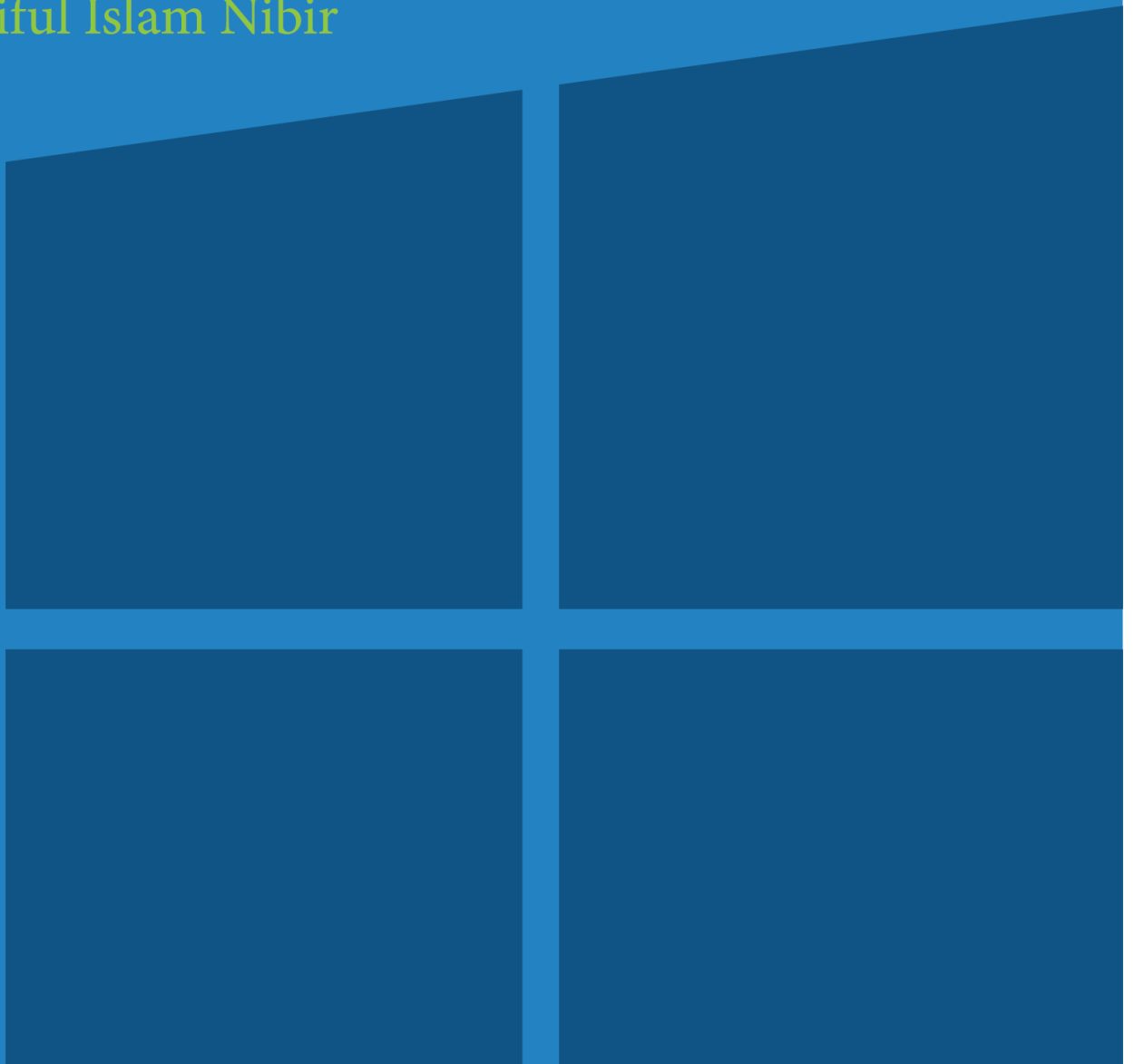


A Complete Guide for Universal Windows Platform Developers

Universal Windows Platform Complete Solution

Rahat Yasir

Shariful Islam Nibir



Universal Windows Platform

Complete Solution

Rahat Yasir

Md. Shariful Islam Nibir

Copyright © 2016 By,

Rahat Yasir | rahat.anindo@live.com

Md. Shariful Islam Nibir | nibirsharif@outlook.com

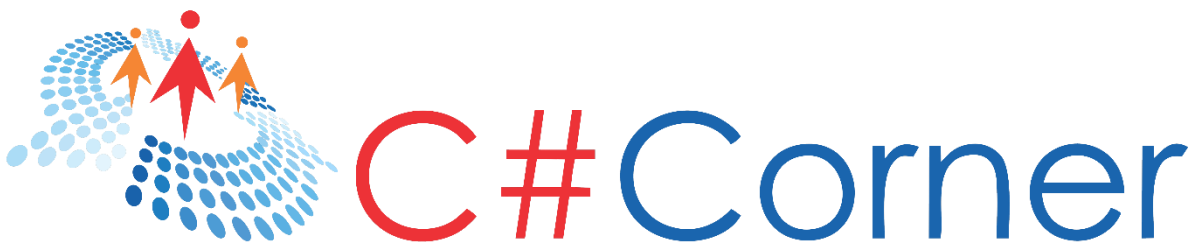
All rights reserved. No part of this book may be used or reproduced in any manner whatsoever without written permission except in the case of brief quotations embodied in critical articles or reviews.

This book is written for the beginners who are planning to develop Universal Windows Platform Application and it would be a gentle introduction of learning Universal Windows Platform App developing to others. It will also help MSPs (Microsoft Student Partners) to give them proper guideline about Universal Windows Application.

This book is dedicated to all the members of .NET Community in Bangladesh.

First Edition: May 2016

This book is exclusively available only on [C# Corner](#).



This is the second eBook of Complete Solution book series.

First eBook of Complete Solution series is “Windows Phone 8.1 Complete Solution”. First eBook of this series is viewed by my more than 15,000 developers and is downloaded more than 5500 times. Authors of “Windows Phone 8.1 Complete Solution” are Rahat Yasir & Md. Shariful Islam Nibir.



About the Authors

Rahat Yasir | @anind0

He is two times Microsoft Most Valuable Professional Award holder. Currently, he is Microsoft MVP in Windows Platform Development category. He has more than 5 years of experience in developing Windows phone / Universal windows platform apps. He is now working as a System Developer in SoftwarePeople | adPeople world wide. He is an Ex Microsoft Student Partner and Youth Spark Advocate. He is the founder of BD Devs, a team of local app developers who develop local essential Windows Phone apps for free. He is also a runner up of Microsoft Imagine Cup 2012, Bangladesh and one of the top 10 image processing app developers of Nokia Future Capture Hackathon, Lund, Sweden. In last three years, more than 400 thousands of developers all over the world followed his blogs and articles. He is one of the authors of the first eBook on Windows phone 8.1 app developments.



Md. Shariful Islam Nibir | @nibirsharif

Md. Shariful Islam Nibir is an enthusiast software developer. He is currently working as a Technical Operations Consultant in Quintiq Sdn Bhd, Malaysia. He has more than 4 years of experience in developing Windows phone / Universal Windows platform apps. He is the co-founder of BD Devs, a team of local app developers who develop local essential Windows Phone apps for free. He is the second runner-up of Microsoft Imagine Cup 2015, Bangladesh round and three times Windows Phone National App-a-thon winner. He provided his valuable inputs for the first eBook on Windows Phone 8.1 app developments.

Table of Contents

1. Introduction
2. IDE Installation
3. Universal Windows Platform Controls Part 1
4. Universal Windows Platform Controls Part 2
5. Universal Windows Platform Controls Part 3
6. Split View
7. List Box
8. XAML Styling
9. App manifest
10. Extended Splash Screen
11. Page Navigation & passing complex object
12. Command Bar
13. Data Binding
14. MVVM
15. Map Control
16. Azure Mobile Service
17. Http Client/Web Client
18. Live Tiles & Push Notification
19. Local Storage
20. PhoneGap Barcode Reader
21. Appendix

Introduction

Universal Windows Platform Complete Solution is the first complete eBook on Universal Windows Platform app development. Authors of this book are Rahat Yasir and Md. Shariful Islam Nibir. Both the authors are experienced Windows phone and Universal Windows platform app developers and have more than 5 years of experience in this field.

Universal Windows Platform Complete Solution book is for those developers who are willing to develop Universal Windows platform apps.

Pre-requisite of this book is object oriented programming in C#.

New Universal Windows Platform developers can also start their app developing career with this book. This book covers both basic and intermediate level Universal Windows platform app development topics.

This book has mentioned different ways of Universal Windows platform app development like developing beautiful, efficient and advanced Universal Windows platform apps using Visual Studio as this is for beginners and intermediate level developers and Universal Windows platform app development using cross platform technology like PhoneGap (Cordova) as this is for cross platform app developers and web developers who are willing to develop non-native windows phone apps.

You can read articles of this book from here,

<http://www.c-sharpcorner.com/members/bd-devs/articles>

You can also follow the blog site of authors,

<http://learnwithbddevs.wordpress.com/>

All the chapters' source code can be [found here](#),

IDE Installation

In this chapter, we'll explain how to install Visual Studio Enterprise 2015 step by step to develop apps for Universal Windows Platform. It's pretty much straight forward and we will show you all the steps one by one, so that you can have the idea on what's going on when you'll install it in your own personal computer.

So let's get crack in Visual Studio Enterprise 2015 IDE Installation process.

Downloading the Visual Studio Enterprise 2015 from MSDN

First of all, download Visual Studio Enterprise 2015. If you have DreamSpark, BizSpark or MSDN account, you can download it from there. Here, I've downloaded it from my MSDN account. After login in your account, you can find all software in subscriber download section and choose any version of Visual Studio from here.

Visual Studio

- [Visual Studio 2015](#)
- [Visual Studio Enterprise 2015](#)
- [Visual Studio Professional 2015](#)
- [Visual Studio Test Professional 2015](#)
- [Visual Studio Express 2015](#)
- [Visual Studio Team Foundation Server 2015](#)
- [Release Management for Visual Studio 2015](#)
- [Other editions and products](#)

Figure 1

Click on any version of Visual Studio and then you will see below section in a new page. Here, you can see Visual Studio Professional 2015 both 32 bit and 64 bit downloading option. You can also claim your product key by clicking on the product keys button.

[Downloads](#) / "Visual Studio Professional 2015" (5 results) Sorted by: [Release Date](#)

Visual Studio Professional 2015 (x86 and x64) - DVD (English) [Product Keys](#) [Download](#)

 | English | Release Date: 7/20/2015 | [Details](#) 3882 MB

Figure 2

After downloading the .iso file, go to the destination folder and open the folder. Inside the folder you can see these files.

| Name | Date modified | Type | Size |
|---------------------|-------------------|-------------------|----------|
| packages | 7/10/2015 7:43 AM | File folder | |
| Standalone Profiler | 7/10/2015 7:43 AM | File folder | |
| autorun | 7/10/2015 6:51 AM | Setup Information | 1 KB |
| vs_enterprise | 7/10/2015 7:43 AM | Application | 2,985 KB |

Figure 3

Start Installation Process

Open the “vs_enterprise” application file and you will see the Windows below one by one.



Figure 4

Then you will see two installation options and they are typical and custom. Select custom installation process and then click “**Next**” button at the bottom.

After that you will see figure 5. Select features that you want to install. Few feature installation may require internet connection to download setup files and install properly. Kindly check internet connection before starting the installation process. At the top, the developer will get Universal Windows App development features. You need to select that option from the feature list. After selecting your desired features, click “**Install**” button.



Figure 5

The installation process will start subsequently. It will take some time based on your computer configuration. High configuration PCs take average one and a half or two hours.

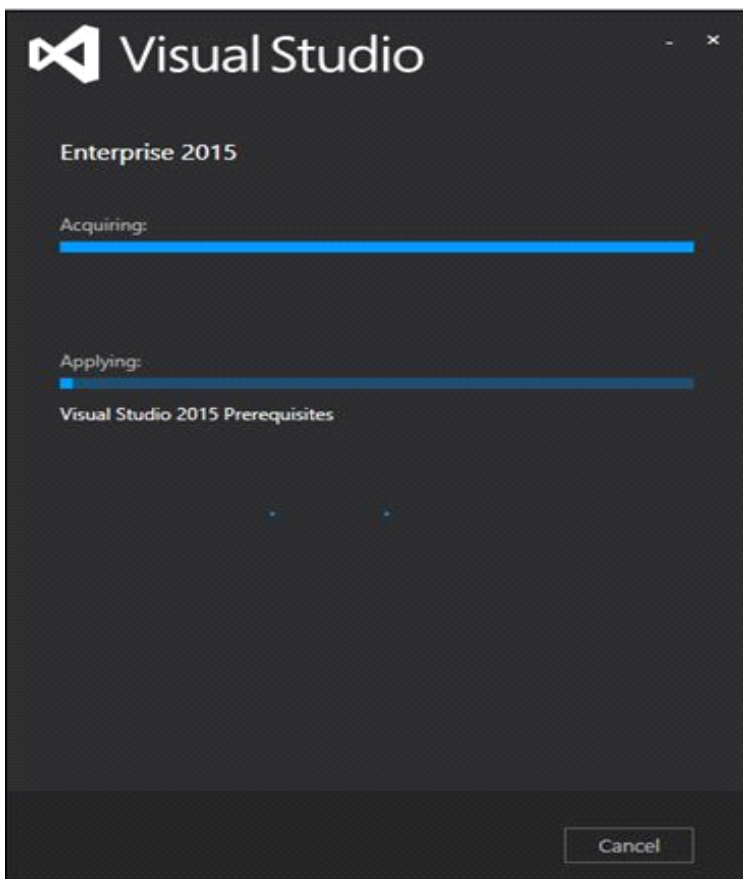


Figure 6

Restarting Your Computer

After completing the installation process, you will see an option to restart your computer to complete the installation process.

Hit the “**Restart Now**” button. It will restart your computer. After some moment, the installation process will be done and the user will get this message window.

Launching Visual Studio Enterprise 2015

Now launch Visual Studio Enterprise 2015 from the installed software list and they you will see below screen.

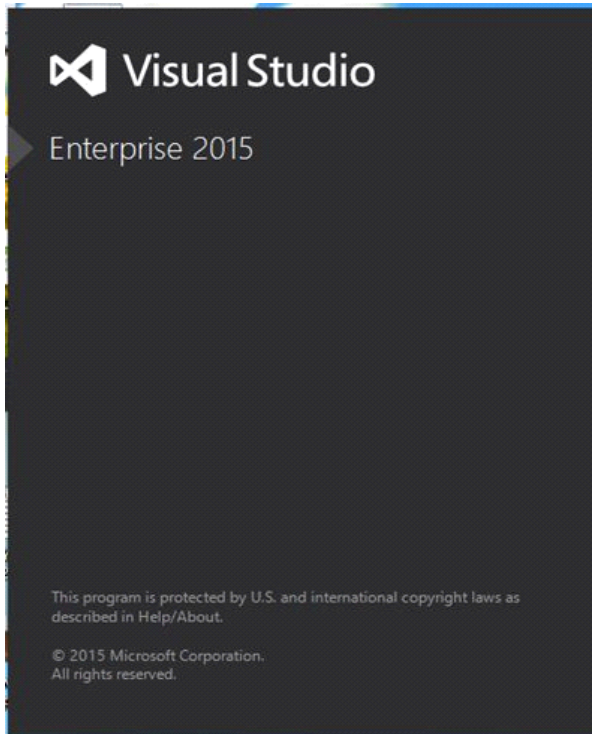


Figure 7

Then you’ll be asked to “**Sign in**” with your Microsoft account, you can do so or just click “**Not now, may be later**”.

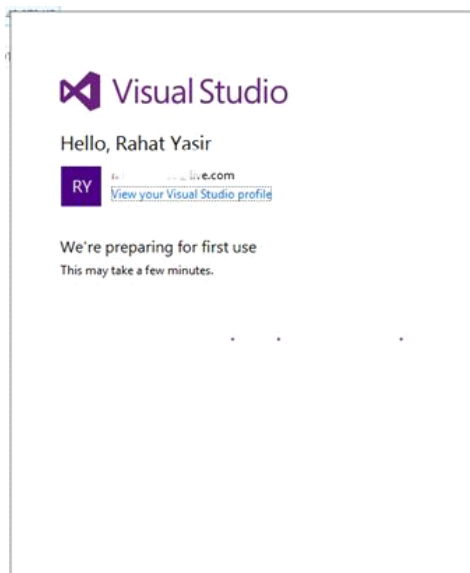


Figure 8

Now Your Visual Studio Enterprise 2015 is ready to use. You will see Visual Studio screen like the one shown below.

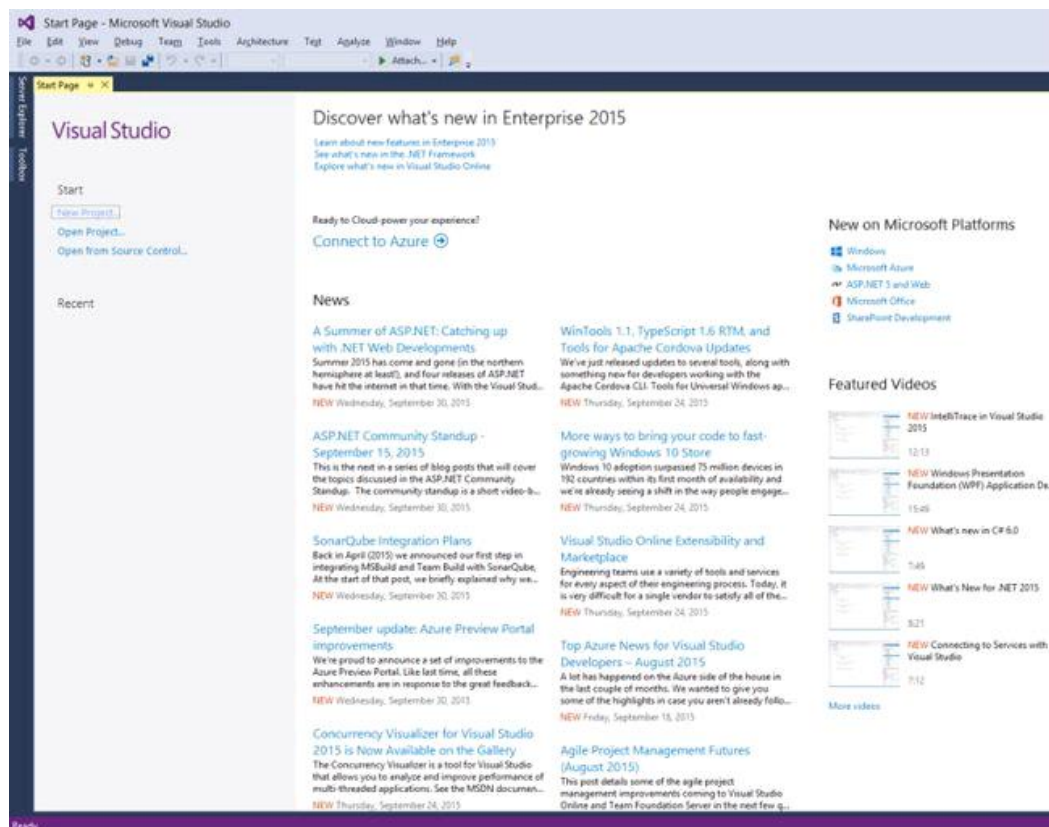


Figure 9

Activating Visual Studio

Now one more thing to do is to activate your Visual Studio with the Product ID which you have got from your MSDN account. Go to Help >> Register Product Key.

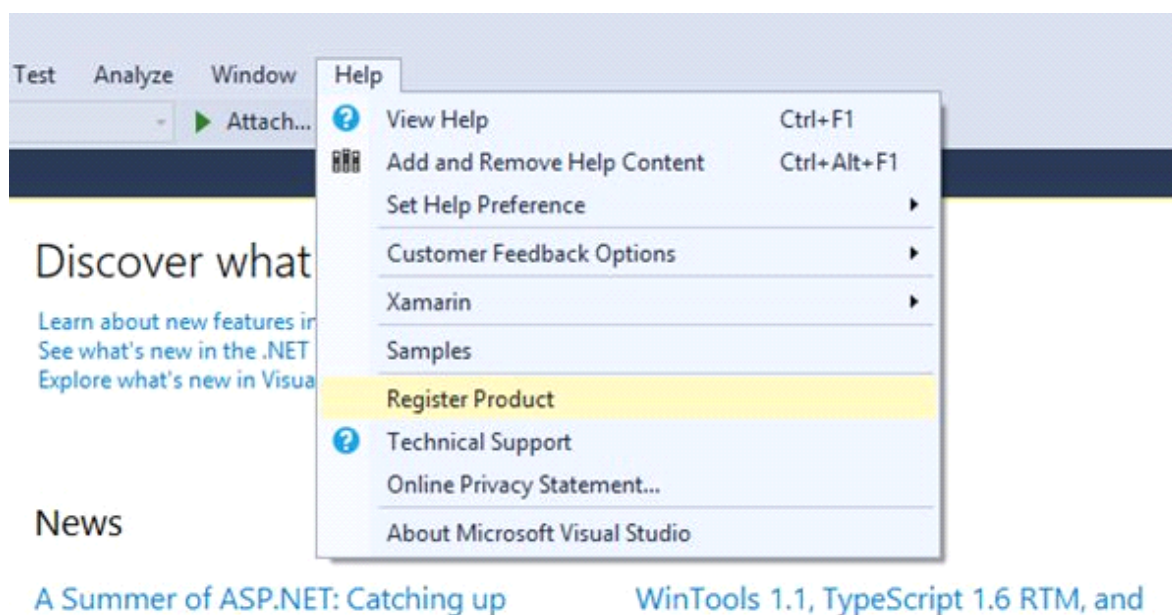


Figure 10

Hit the Register Product Key and you will see the window as shown below. Click the “**Licence with a Product Key**” hyperlink.

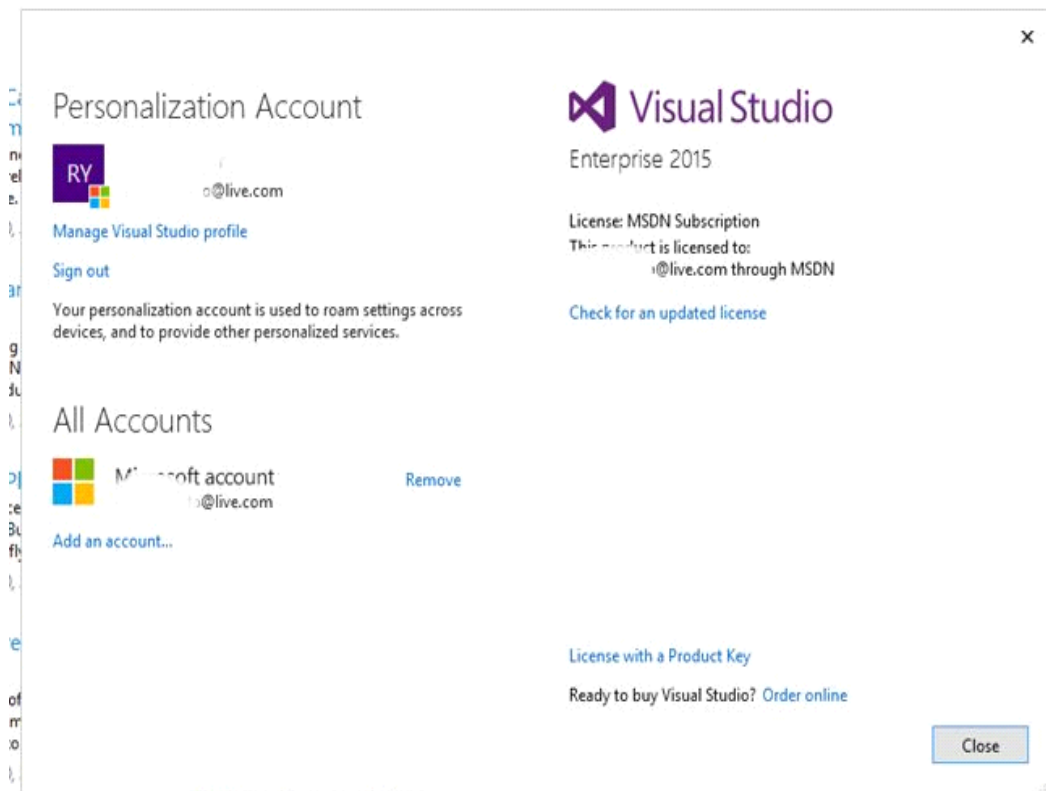


Figure 11

Input your product key on the below product key text box Subsequent to it, Hit “Apply” button and you’re ready to go.

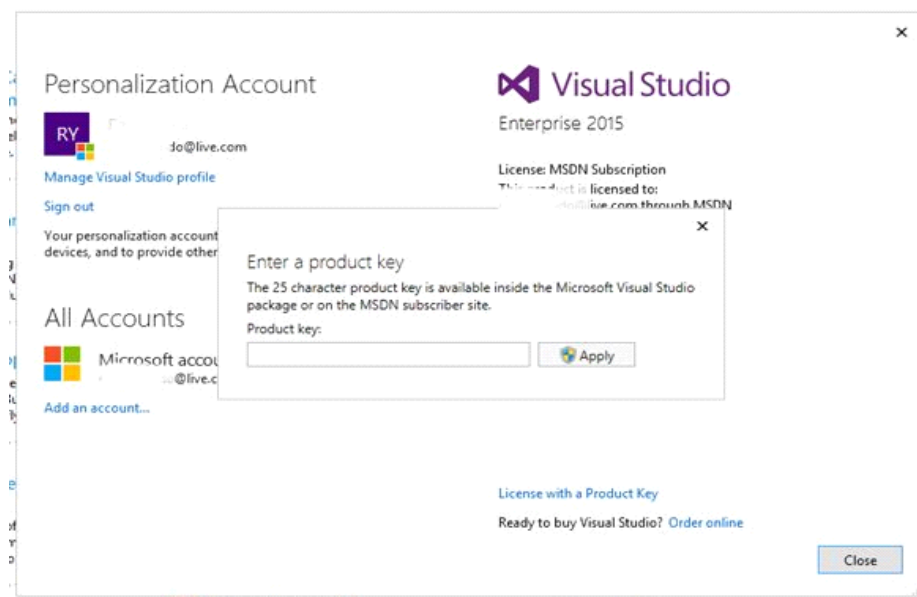


Figure 12

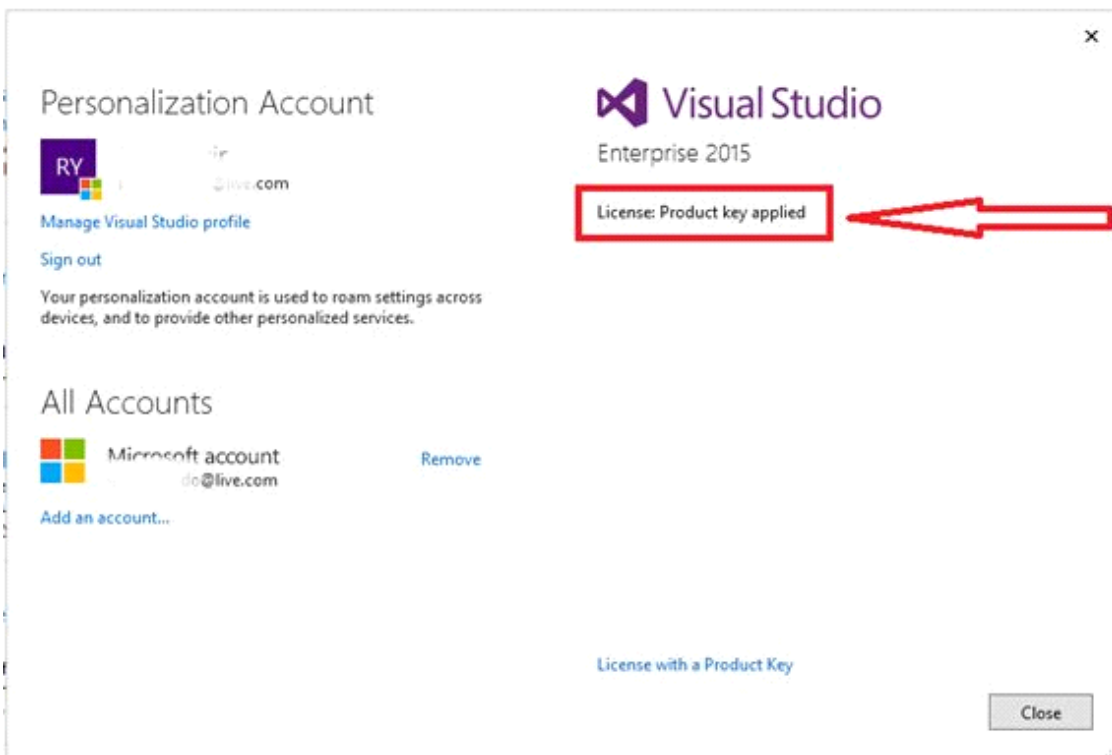


Figure 13

Get Ready to Use Visual Studio

Now, you can make your favorite Universal Windows application. Just click the “**New Project**”. Select Universal app development template and rock on.

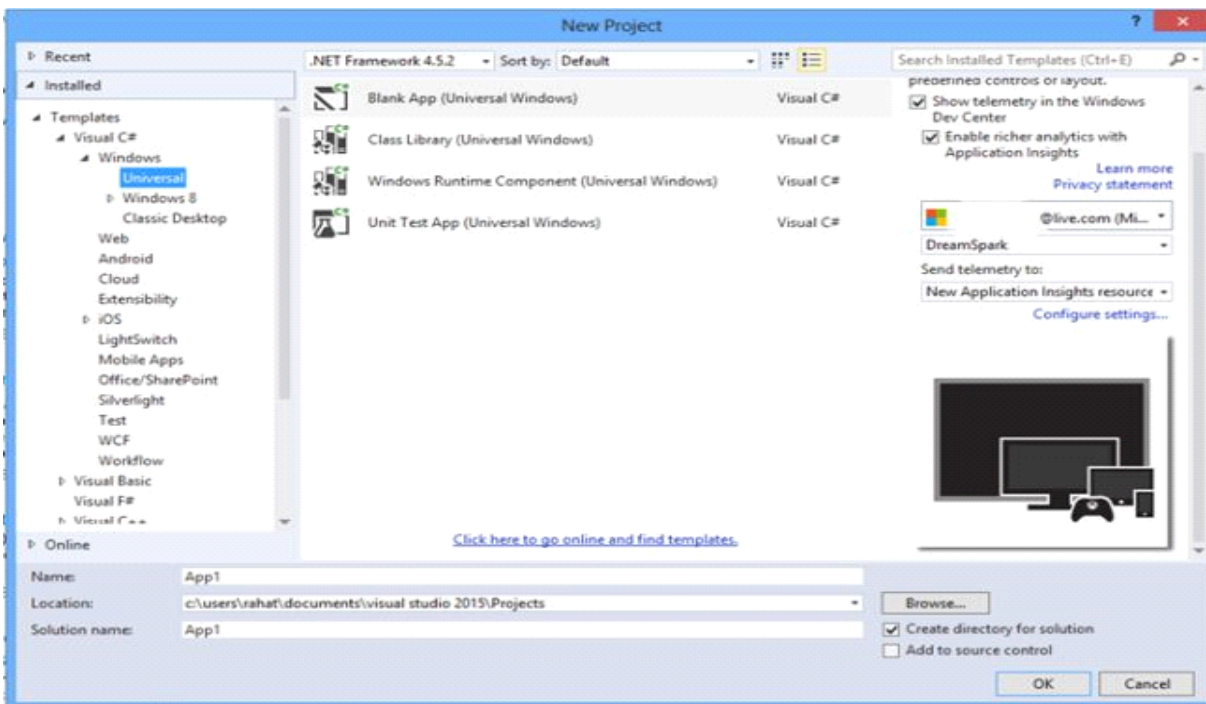


Figure 14

Summary

So, that's it. Start developing awesome applications with the greatest tool of Microsoft. Thanks for reading this article.

Universal Windows Platform Controls – Part 1

In this chapter, we will talk about some basic controls of Universal Windows Platform and XAML is the main design language of Universal Windows Platform. We think, it will help you to understand the basic principle of XAML and how it works. Its attributes and uses are also taught. So let's learn about the essential XAML controls for Universal Windows Platform.

XAML makes your life much easier. Before the existence of XAML, one needs to design lot of pages with same alignment. Moreover, its hectic & frustrating also not an easy task too. It is XAML that brings you the flexibility to make your design portable and easy to arrange. You can copy paste your design and reuse wherever you want. Not only this, you can shape your design whatever you like to do and the power is now in your hand. Now let's see what kind of simple controls you can use in your Universal Windows Platform Application.

Creating a new project

First of all, we will create a project. Open Visual Studio and open a “New Project”. Select “Blank App” and name it “Controls1”.

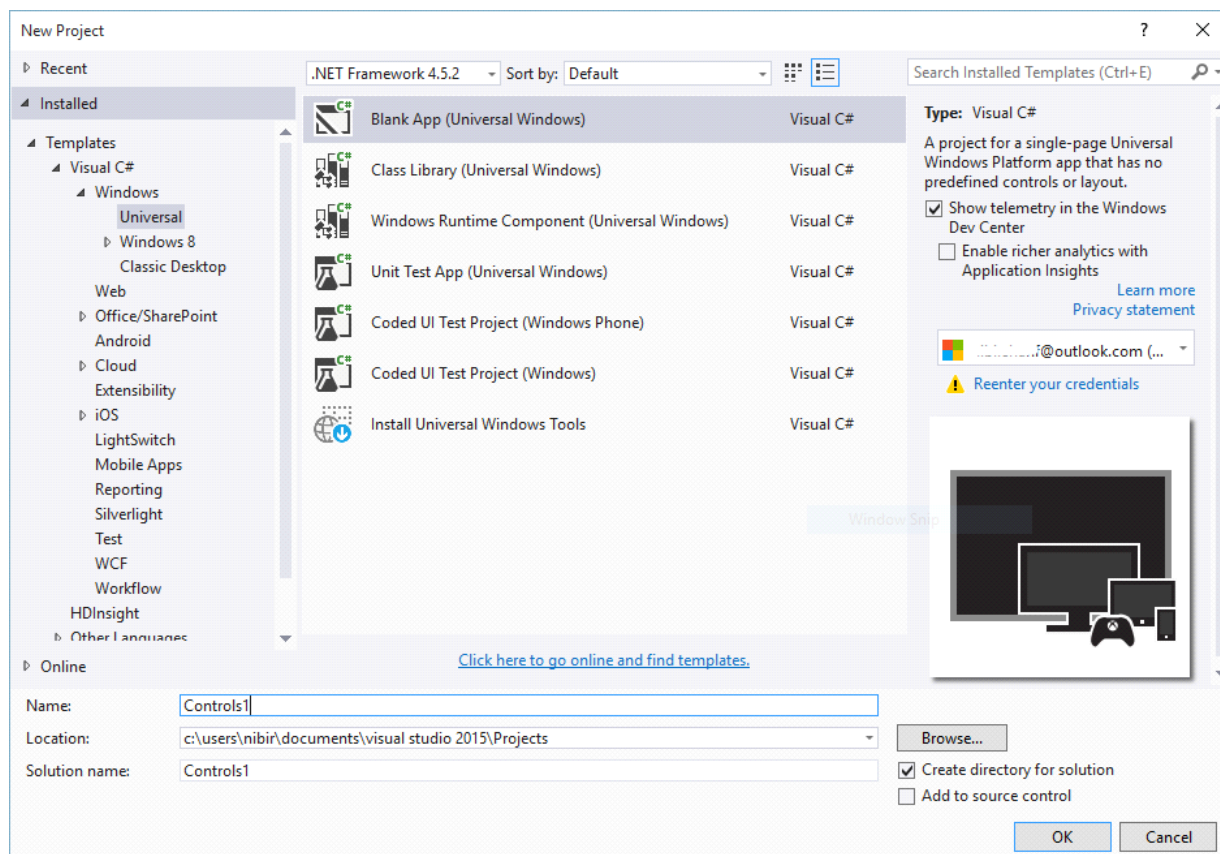


Figure 1

Click OK and you can see Visual Studio like below.

Working with HyperlinkButton

Previously, we have worked with simple Button controls in our “Hello world” application. Now, we will work with another Button like control called “HyperLinkButton”. It's used to link a uniform resource locator popularly known as URL or some other things you like.

To do that, you can see Toolbox in the left side of the Visual Studio and you can find it in “All XAML Controls” section. Click on this control and drag it to you design. Like this,

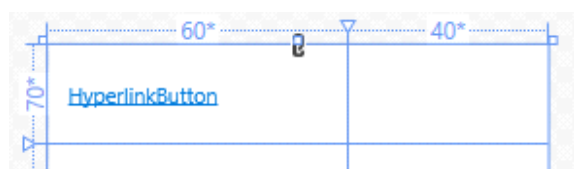


Figure 3

Now, to make sure it works, we need TextBlock control. To do so, drag it from the Toolbox and put it below the Hyperlink button. The designing code is given below.

```
<!--Hyperlink Button-->
<HyperlinkButtonx:Name="HyperlinkButton_1"
    Content="HyperlinkButton"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Margin="10,10,0,0"
    Click="HyperlinkButton_1_Click"
    Grid.Column="0"
    Grid.Row="0"
    Width="114"
    Height="50"/>

<TextBlockx:Name="HB_TextBlock"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    TextWrapping="Wrap"
    Height="50"
    Width="124"
    FontSize="24"
    Grid.Column="1"
    Grid.Row="0"
    Margin="10"/>
```

Listing: 1

In HyperlinkButton, we have an event controller named HyperlinkButton_1_Click. It creates a code block which will handle the background task, when you will click in the hyper link Button and we will show a confirmation message in the TextBlock named HB_TextBlock.

Making Grid

We have made some Grids in our main Grid to arrange the controls in these Grids like Grid 1, 2 and so on. You can make Grids wherever you want, you can customize the Grid as per your requirements.

```
<Grid Background="{ ThemeResourceApplicationPageBackgroundThemeBrush}">

    <Grid.RowDefinitions>

        <RowDefinition Height="70*" />

        <RowDefinition Height="80*" />

        <RowDefinition Height="68*" />

        <RowDefinition Height="93*" />

        <RowDefinition Height="145*" />

        <RowDefinition Height="124*" />

        <RowDefinition Height="60*" />
```

```

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

    <ColumnDefinition Width="60*" />

    <ColumnDefinition Width="40*" />

</Grid.ColumnDefinitions>

</Grid>

```

Listing: 2

Now, the main Grid will look like this.

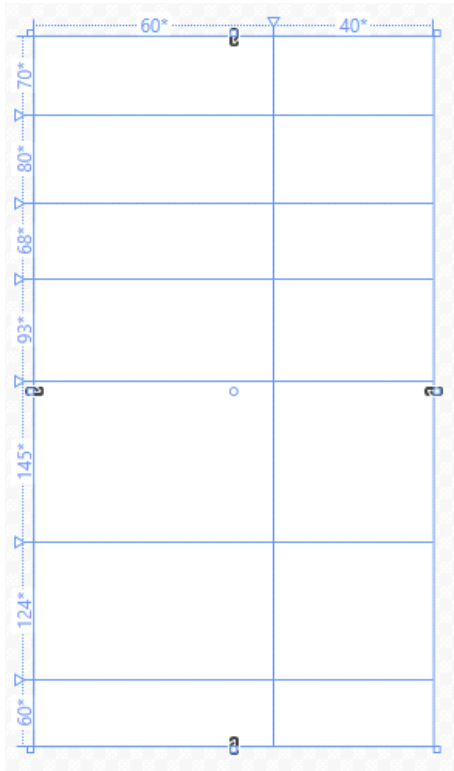


Figure 4

You can define Row Definitions which states number of rows along with their width and Column Definitions according to their width. We have seven rows and two columns here.

Now, open MainPage.xaml.cs and put the code below the constructor.

```

privatevoid HyperlinkButton_1_Click(object sender, RoutedEventArgs e)
{
    HB_TextBlock.Text = "OK";
}

```

Listing 2

Now run the application and it will look like the picture below after you will click in the HyperlinkButton.

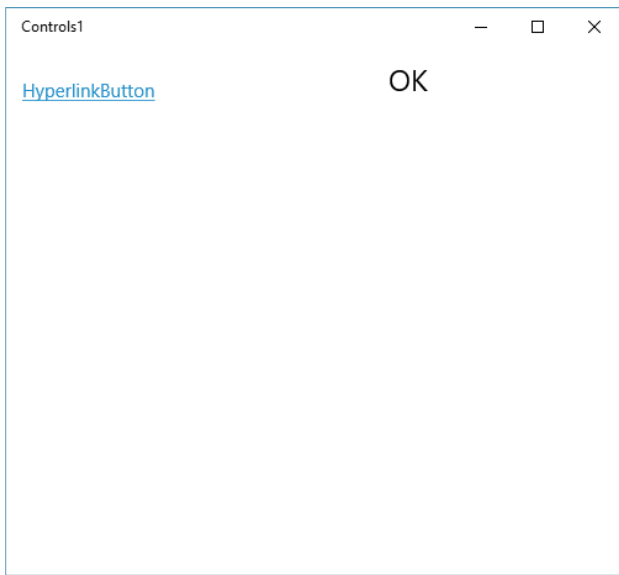


Figure 5

Working with RadioButton

Well if you can do that, then you are on move. Now, we will take another control named RadioButton. To use it, we can drag it from TextBox, put it in another Grid and also a TextBox in Row 1. The customized code will look like this or you can simply drag a control and test separately as it entirely depends on you. I suggest you to do as I do.

So, our design will look like this,

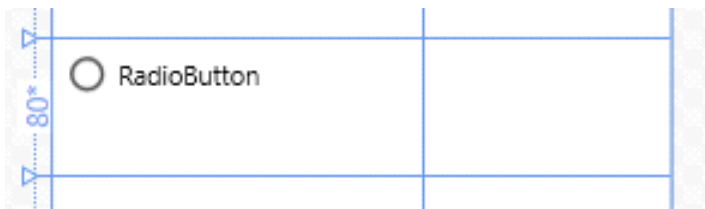


Figure 6

And the designing code is below.

```
<RadioButtonx:Name="RadioButton_1"
Content="RadioButton"
HorizontalAlignment="Left"
Margin="10,4,0,0"
Grid.Row="1"
```



```
Grid.Column="0"
VerticalAlignment="Top"
Checked="RadioButton_1_Checked" Height="66" Width="114"/>
```

```
<TextBlockx:Name="RB_TextBlock"
HorizontalAlignment="Left"
VerticalAlignment="Top"
Margin="10,10,0,0"
TextWrapping="Wrap"
Height="60"
Width="124"
FontSize="24"
Grid.Column="1"
Grid.Row="1"/>
```

Listing 3

In our RadioButton, we have an event handler named RadioButton_1_Checked and in our event handler, we will show the confirmation message whether it's checked or unchecked.

```
private void RadioButton_1_Checked(object sender, RoutedEventArgs e) {
    if (RadioButton_1.IsChecked == true) {
        RB_TextBlock.Text = "Checked";
    }
    Else {
        RB_TextBlock.Text = "Not checked";
    }
}
```

Listing 4

Here, we are checking whether our RadioButton is checked or not, if it's checked (true), the TextBlock will show "Checked" or if it's unchecked (false), the TextBox will show "Not checked".

After you run your application, it will look exactly like this.

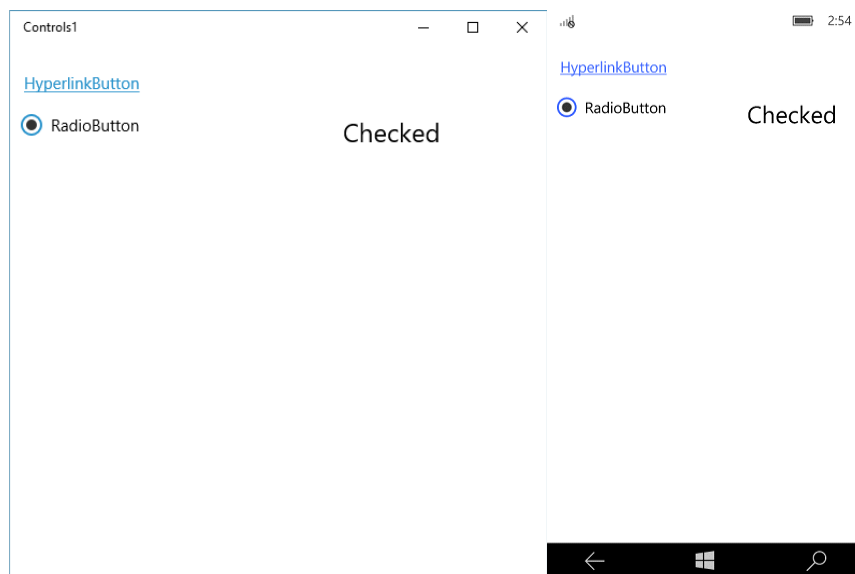


Figure 7

Working with TextBlock

Another control, we use mostly in our every application is TextBlock. We have used it in our previous controls also. We will show static data in our TextBlock.x, “Hello world”.

The design will look like this.



Figure 8

Designing code is below.

```
<!--Text Block-->
<TextBlock Text="Hello world"
    HorizontalAlignment="Left"
    Margin="10,10,0,0"
    Grid.Row="2"
    TextWrapping="Wrap"
    VerticalAlignment="Top"
    Height="40"
    Width="380"
    FontSize="24" Grid.ColumnSpan="2"/>
```

Listing 5

We don't need any Button or event handler in this case because the text is given statically in the design (Text="Hello world").

After you run your application, it will look exactly like this.

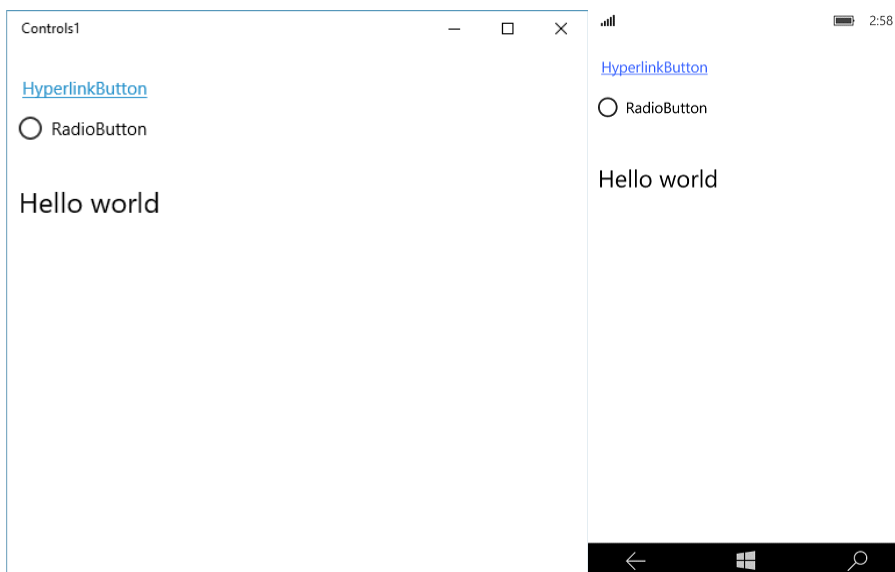


Figure 9

Working with ToggleSwitch

Another control, we will talk about is ToggleSwitch. It's really a good control as it will make your application cooler than before. I think you know how to use a control now as we have done it before. So, just take this control and take another TextBlock and the design will look like this.



Figure 10

The designing code is below,

```

<!--Toggle Switch-->
<ToggleSwitchx:Name="ToggleSwitch_1"
    Header="ToggleSwitch"
    Margin="10,10,0,0"
    Grid.Row="3"
    VerticalAlignment="Top"
    Toggled="ToggleSwitch_1_Toggled"
    Width="196"
    Height="73"/>

<TextBlockx:Name="TS_TextBlock"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Margin="10,10,0,0"
    TextWrapping="Wrap"
    Height="73"
    Width="124"
    FontSize="24"
    Grid.Column="1"
    Grid.Row="3"/>

```

Listing 6

We have an event handler here, so the C# code is given below.

```

privatevoid ToggleSwitch_1_Toggled(object sender, RoutedEventArgs e) {
    if (ToggleSwitch_1.IsOn == true) {
        TS_TextBlock.Text = "This is On";
    }
    else {
        TS_TextBlock.Text = "This is Off";
    }
}

```

Listing 7

We applied the same logic here like that in RadioButton.

After you run your application, it'll look exactly like this.

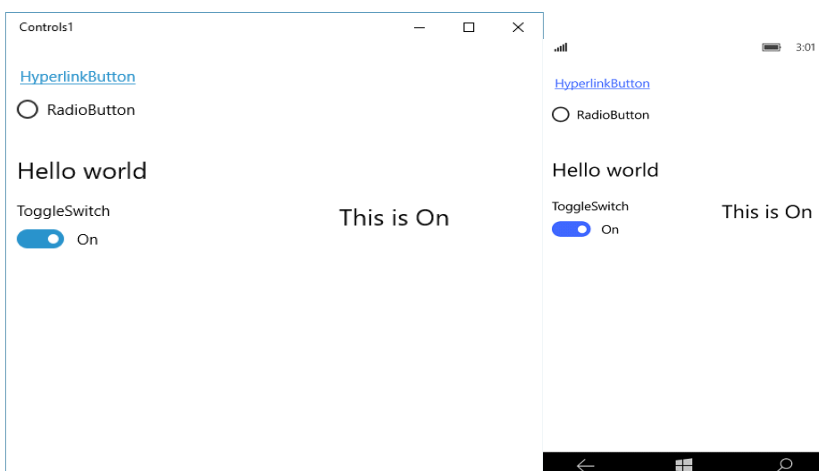


Figure 11

Working with ListBox

Our fifth control will be ListBox which is data binding control. It's an important control which has some complicated structure. So let's see how, we can use it in our application.

Like other controls, drag it from Toolbox and put in the Grid. Here, we need a Button and TextBlock controls.

The design will look like this,

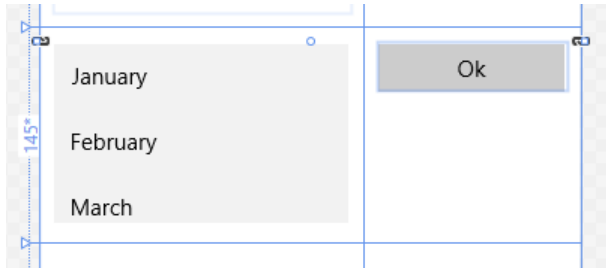


Figure 12

The designing code is given below,

```
<!--List Box-->
<ListBoxx:Name="ListBox_1"
    HorizontalAlignment="Left"
    Height="120"
    Margin="10,10,0,0"
    Grid.Row="4"
    VerticalAlignment="Top"
    Width="196"
    ScrollViewer.VerticalScrollBarVisibility="Visible">
<ListBoxItem Content="January"/>
<ListBoxItem Content="February"/>
<ListBoxItem Content="March"/>
<ListBoxItem Content="April"/>
<ListBoxItem Content="May"/>
<ListBoxItem Content="June"/>
<ListBoxItem Content="July"/>
<ListBoxItem Content="August"/>
<ListBoxItem Content="September"/>
<ListBoxItem Content="October"/>
<ListBoxItem Content="November"/>
<ListBoxItem Content="December"/>
</ListBox>

<Button Content="Ok"
    x:Name="Ok"
    Grid.Column="1"
    HorizontalAlignment="Left"
    Margin="10,10,0,0"
    Grid.Row="4"
    VerticalAlignment="Top"
    Width="110"
    Click="Ok_Click"/>

<TextBlockx:Name="LB_TextBlock"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Margin="10,53,0,0"
    TextWrapping="Wrap"
    Height="82"
    Width="124"
    FontSize="24"
    Grid.Column="1"
    Grid.Row="4"/>
```

Listing 8

Here, we have an event handler named “Ok_Click” and we have binded some month’s name inside the ListBox’s starting and closing tags. TextBlock’s name is LB_TextBlock. Hence, the C# code will look like this.

```
private void Ok_Click(object sender, RoutedEventArgs e)
{
    string[] month = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October",
    "November", "December" };
    if (ListBox_1.SelectedValue != null)
    {
        LB_TextBlock.Text = month[ListBox_1.SelectedIndex];
    }
    else
    {
        LB_TextBlock.Text = "Select a item from list.";
    }
}
```

Listing 9

Here, we have created a string Array named month and the array index’s values are the month’s name. In <If decision statement>, first we’re checking if the ListBox is selected or not, if an item is selected, we’re matching the SelectedIndex’s value with our array Index’s value and if no item is selected then an alert message will be shown in the TextBlock.

If we run the application, it will look exactly like this,

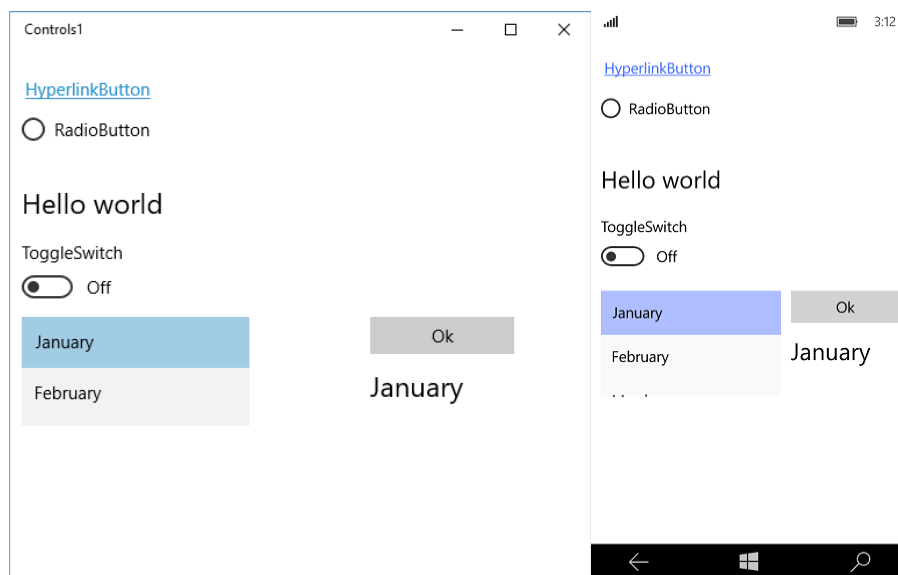


Figure 13

Working with ComboBox

Now, we will talk about a similar control and it’s really awesome compared to ListBox, just works exactly same as ListBox but it depends on your application which will be more appropriate in case of your requirements. It is called ComboBox. Take it from ToolBox or you can just write XAML on your own. Therefore, the design will look like this,

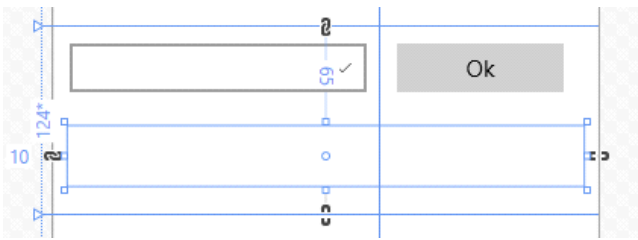


Figure 14

The designing code is given below,

```

<!--Combo Box-->
<ComboBox:Name="ComboBox_1"
  HorizontalAlignment="Left"
  Margin="10,0.167,0,0"
  Grid.Row="5"
  VerticalAlignment="Top"
  Width="220">
<ComboBoxItem Content="January"/>
<ComboBoxItem Content="February"/>
<ComboBoxItem Content="March"/>
<ComboBoxItem Content="April"/>
<ComboBoxItem Content="May"/>
<ComboBoxItem Content="June"/>
<ComboBoxItem Content="July"/>
<ComboBoxItem Content="August"/>
<ComboBoxItem Content="September"/>
<ComboBoxItem Content="October"/>
<ComboBoxItem Content="November"/>
<ComboBoxItem Content="December"/>
</ComboBox>

<TextBlockx:Name="CB_TextBlock"
  HorizontalAlignment="Left"
  VerticalAlignment="Top"
  Margin="10,65.167,0,0"
  TextWrapping="Wrap"
  Height="40"
  Width="380"
  FontSize="24"
  Grid.Row="5"Grid.ColumnSpan="2"/>

<Button Content="Ok"
  x:Name="Ok_1"
  Grid.Column="1"
  HorizontalAlignment="Left"
  Margin="10,0.167,0,0"
  Grid.Row="5"
  VerticalAlignment="Top"
  Width="125"
  Click="Ok_1_Click"/>

```

Listing 10

And the C# code is here.

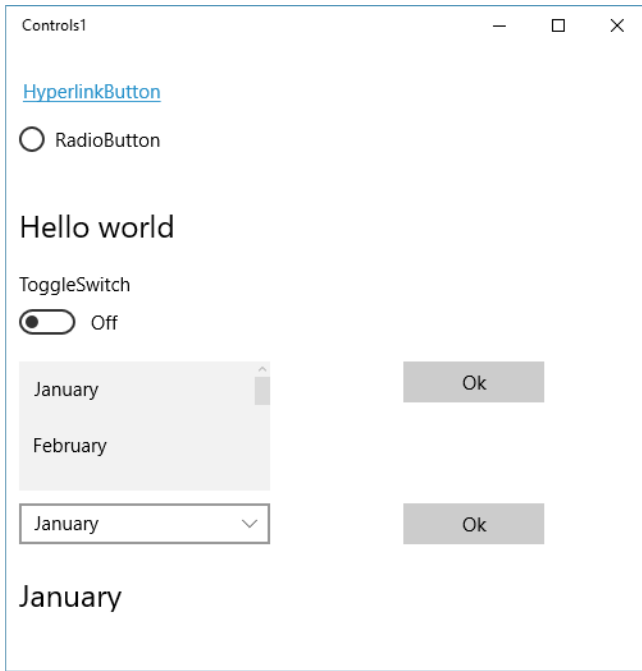
```

private void Ok_1_Click(object sender, RoutedEventArgs e)
{
    string[] month = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October",
    "November", "December" };
    if (ComboBox_1.SelectedValue != null)
    {
        CB_TextBlock.Text = month[ComboBox_1.SelectedIndex];
    }
    else
    {
        CB_TextBlock.Text = "Select a item from list.";
    }
}

```

```
}  
}
```

Listing 11



If we run the application, it will look exactly like this.

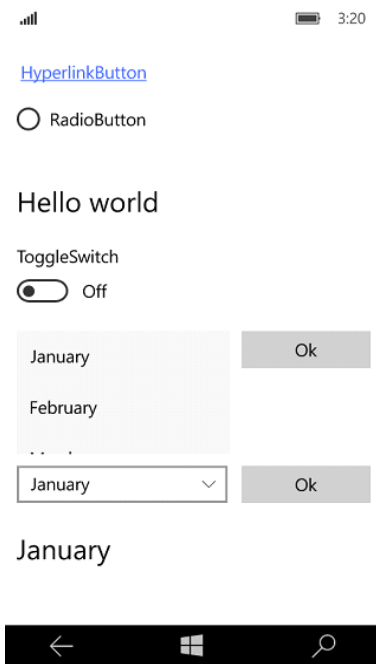


Figure 15

Adding a User Control

Lastly, we will talk about Popup Box with a Button control and it will show some messages. For this, we need a User Control. Go to the Solution Explorer and Add >> New Item.

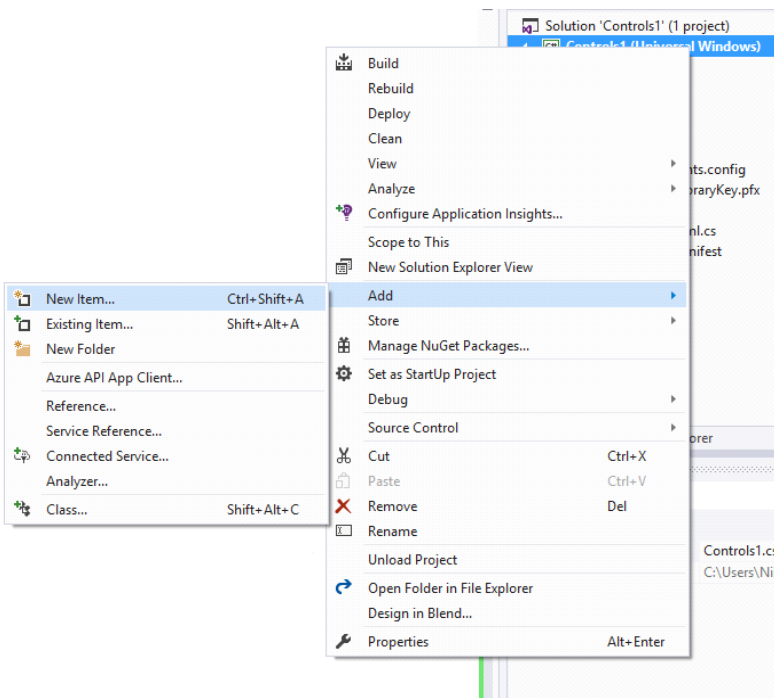


Figure 16

Now you have to select User Control and give it a name called "PopupPanel".

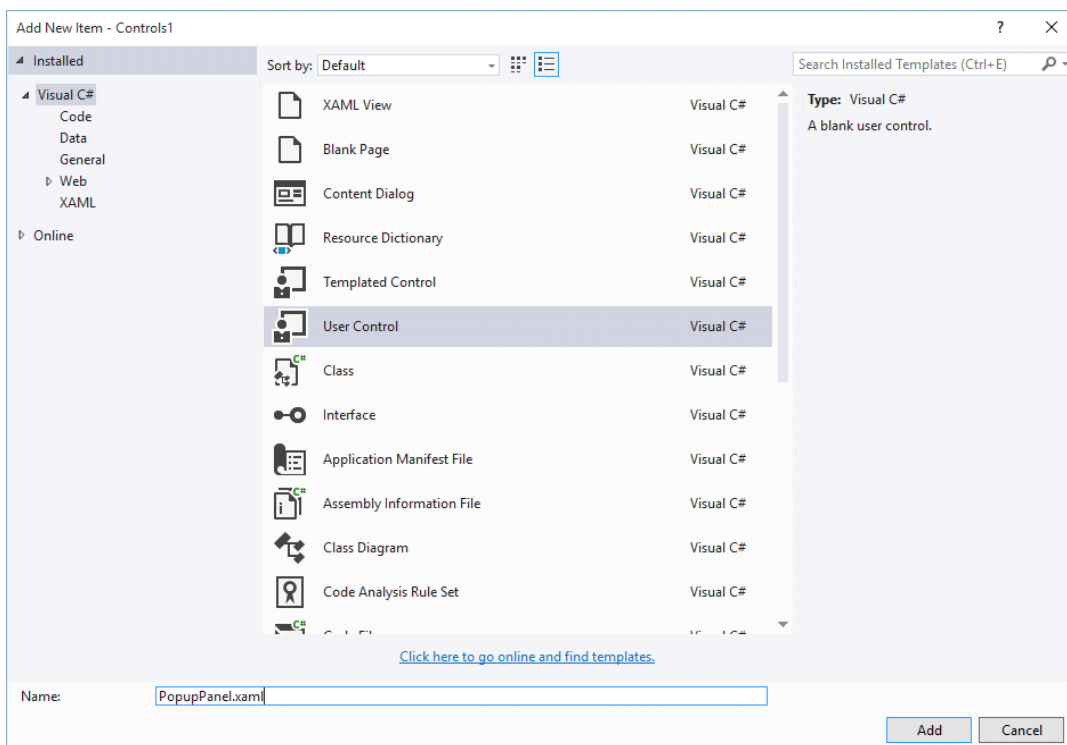


Figure 17

Customize the XAML code, mainly the Grid section.

<Grid>

```
<Border BorderBrush="{StaticResource ApplicationForegroundThemeBrush}" BorderThickness="1"
```

```
Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
```

```
<StackPanel Orientation="Vertical" Height="200" Width="200" VerticalAlignment="Center">
```



```

<TextBlock Text="This is a Popup!" VerticalAlignment="Center" HorizontalAlignment="Center"
Margin="0,60,0,0" />

<TextBlock Text="Hit the button again to hide me" VerticalAlignment="Center"
HorizontalAlignment="Center" Margin="0,10,0,0" TextWrapping="Wrap" TextAlignment="Center"
/>

<ButtonHorizontalAlignment="Center" Content="Close Popup" Click="ClosePopup" />
</StackPanel>

```

```
</Border>
```

```
</Grid>
```

Listing 12

Here, we have Border brush, StackPanel which will be bounded by the TextBlocks and a Button. The design will look like the one shown below-

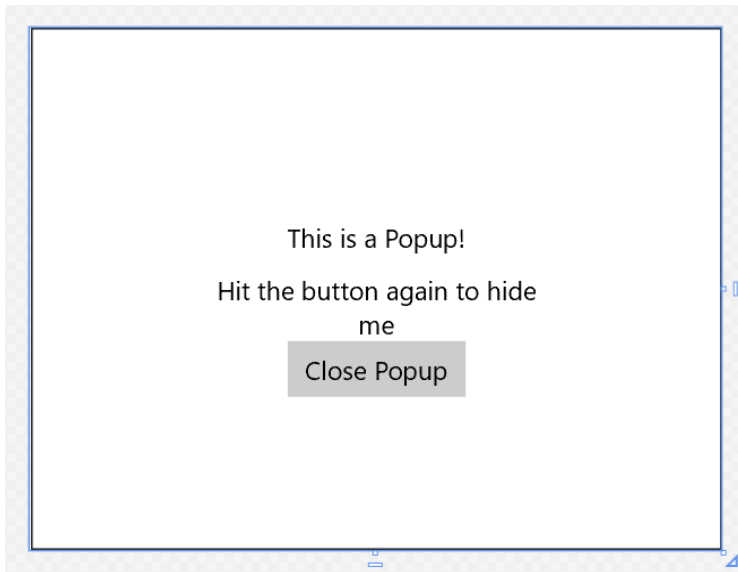


Figure 18

The C# code of PopupPanel.xaml.cs is given below. It's mainly the Button's event handler.

```

private void ClosePopup(object sender, RoutedEventArgs e)
{
    Popup hostPopup = this.Parent as Popup;
    hostPopup.IsOpen = false;
}

```

Listing 13

We just make our first User Control. It's really helpful when you need a custom control in your application.

Working with Popup Window

Now, in our MainPage.xaml, we have to take a TextBlock which will have a header message called "Popup Window" and a Button with content "Show Popup". The design will look like this,



Figure 19

The designing code is given below,

```
<!--Popup Window-->
<TextBlockHorizontalAlignment="Left"
    Text="Popup Winodow"
    VerticalAlignment="Top"
    Margin="10,10,0,0"
    TextWrapping="Wrap"
    Height="40"
    Width="220"
    FontSize="24"
    Grid.Row="6"/>

<Button Content="Show Popup"
    x:Name="PopupButton"
    Grid.Column="1"
    HorizontalAlignment="Left"
    Margin="10,0,0,0"
    Grid.Row="6"
    VerticalAlignment="Top"
    Width="140"
    Click="PopupButton_Click"/>
```

Listing 14

Our event handler C# code behind is also given here,

```
privatevoidPopupButton_Click(object sender, RoutedEventArgs e)
{
    if (!popup.IsOpen)
    {
        popup.Child = newPopupPanel();
        popup.VerticalOffset = 250.0;
        popup.HorizontalOffset = 100.0;
        popup.IsOpen = true;
    }
}Popuppopup = newPopup();
```

Listing 15

Here, we have created new object of Popup window and checked it in our event handler code block by If decision statement. We have created a Popup Child object, set its position and made it Open equal to true, so that it shows up when it's called.

If we run the application, it'll look exactly like this.

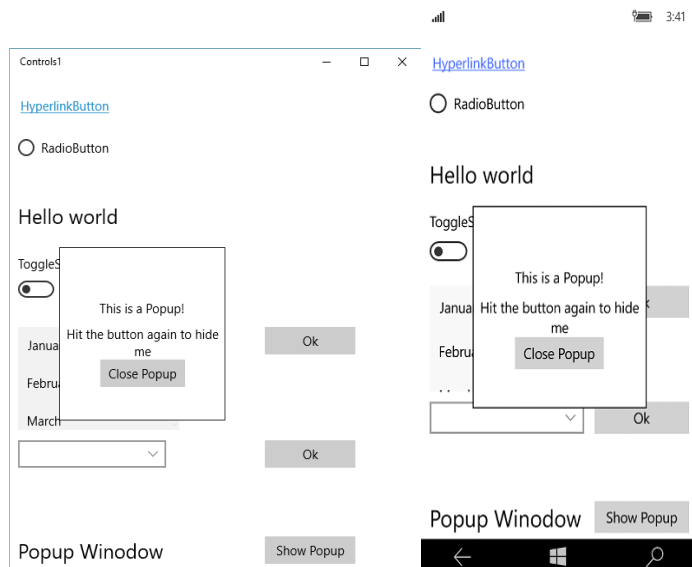


Figure 20

Finalizing and running our Control Application

In the end, our full design will look like the picture below,

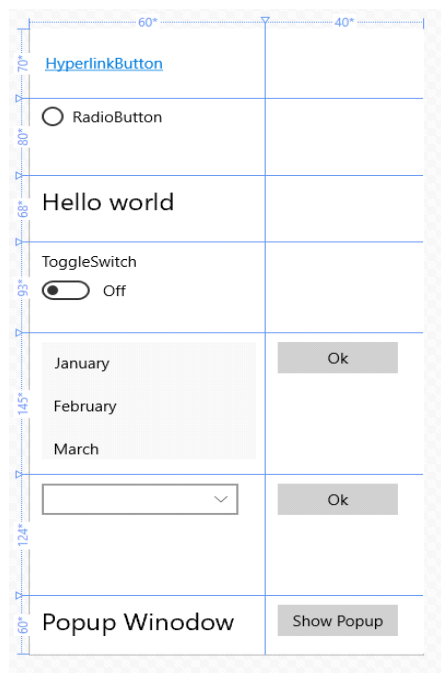


Figure 21

If we run the complete application, it will look exactly like this.

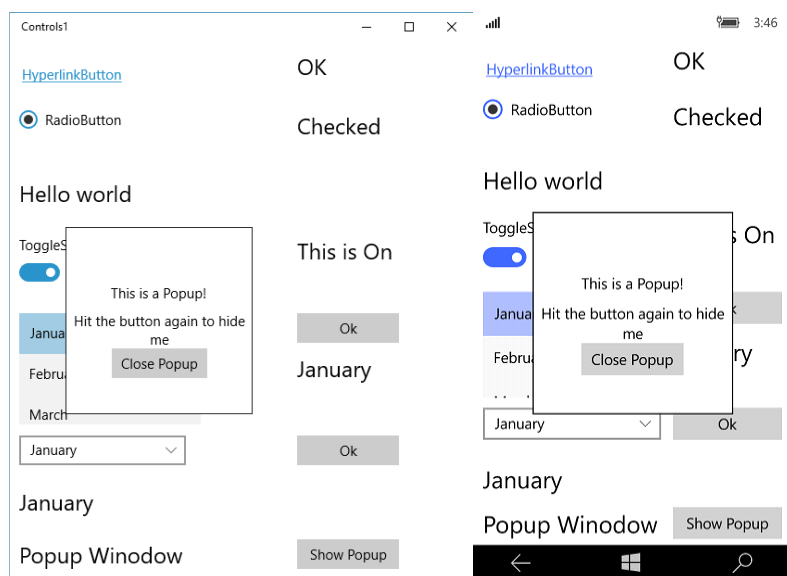


Figure 22

We have talked about seven different controls and their uses. Hopefully, it will give you a little idea of how to use controls and modify with XAML in Universal Windows Platform Application.

Universal Windows Platform Controls – Part 2

In Universal Windows Platform Controls - Part 1, we have seen seven essential controls of Universal Windows Platform. In this, we will learn about more common controls like Input controls. We will mostly talk about TextBox, Date and Time Picker controls. So let's get started.

Working with TextBox Control

First of all, today I'm not going to explain, how to open up project from start. We have done this before. So, I am moving up to the content. Take two TextBlocks and change the Text content to "First Name" & "Last Name". Then take two TextBoxs, name it "firstNameTextBox" & "lastNameTextBox". Take another two TextBlock and name it "welcomeTextBlock" & "nameTextBlock". Arrange them like the one in the picture below.

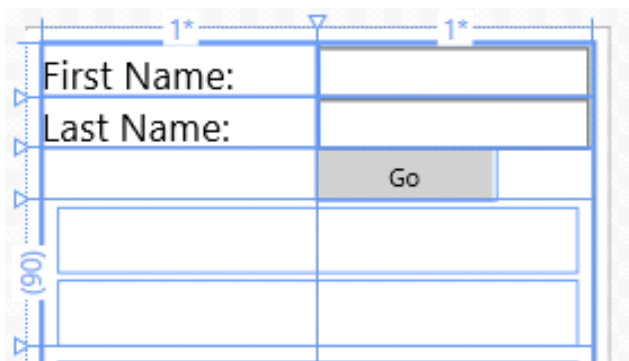


Figure 1

Designing code is also given here.

```
<TextBlock Text="First Name: "FontSize="24"/>
<TextBoxx:Name="firstNameTextBox"Grid.Column="1" />

<TextBlock Text="Last Name: "Grid.Row="1"FontSize="24"/>
<TextBoxx:Name="lastNameTextBox"Grid.Row="1"Grid.Column="1" />

<Buttonx:Name="GoButton"Grid.Column="1"Grid.Row="2" Content="Go"VerticalAlignment="Bottom" Width="110"
Click="GoButton_Click" />

<TextBlockx:Name="welcomeTextBlock"HorizontalAlignment="Left"
Margin="10,5,0,0"Grid.Row="3"TextWrapping="Wrap"VerticalAlignment="Top" Height="40"
Width="320"FontSize="20"Grid.ColumnSpan="2"/>
<TextBlockx:Name="nameTextBlock"HorizontalAlignment="Left" Margin="10,50,0,0"Grid.Row="3"TextWrapping="Wrap"
Width="320"FontSize="20"Grid.ColumnSpan="2" Height="40"VerticalAlignment="Top"/>
```

Listing 1

As we have to handle the Input operation, we used a Button control in the code above and have a click event "GoButton_Click". So our C# code will look like this.

```
privatevoidGoButton_Click(object sender, RoutedEventArgs e)
{
    if (lastNameTextBox.Text != string.Empty)
    {
        if (firstNameTextBox.Text != string.Empty)
        {
            welcomeTextBlock.Text = "Hello,";
            nameTextBlock.Text = firstNameTextBox.Text + " " + lastNameTextBox.Text;
        }
    }
}
```

```

    }
    else
    {
        welcomeTextBlock.Text = "Hello,";
        nameTextBlock.Text = lastNameTextBox.Text;
    }
}
else
{
    welcomeTextBlock.Text = "Sorry,";
    nameTextBlock.Text = "Last name can't be empty!";
}
}

```

Listing 2

Now, what I actually did, is checking the text of “lastNameTextBox” whether it’s empty or not. If it’s not empty, it will be good to go. Then we’re checking the text of “firstNameTextBox”, and if it’s not empty we will do the operation below the second. So, in this case we make a welcome text “Hello” and put the first and last name in the “nameTextBlock” or we will just put the last name in this field.

Otherwise, we’ll give an error message if the last name field is empty, because last name can’t be empty.

Working with Date & Time Picker Control

Now, we will talk about Date and Time Picker controls. Drag and drop or just write your own customized XAML. I like to write my preferable customized XAML. I have taken one Textblock as header to show some text, one DatePicker and one TimePicker and a Button. On the right side, I have also taken a TextBlock as a header field and two other TextBlock to show the date and time you’ll pick. The design is given here.

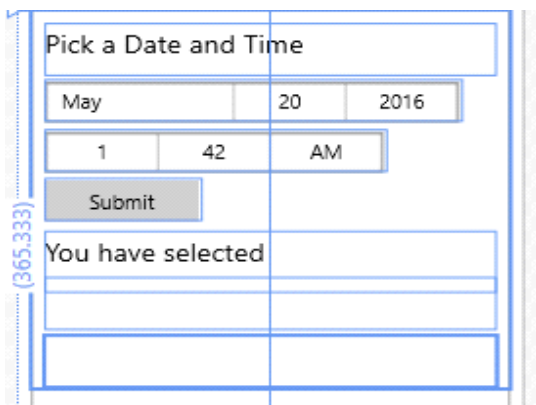


Figure 2

Designing code is given below.

```

<TextBlockHorizontalAlignment="Left" Margin="10,10.333,0,0" Grid.Row="4" TextWrapping="Wrap" Text="Pick a Date and Time" FontSize="20" VerticalAlignment="Top" Height="40" Width="320" Grid.ColumnSpan="2" />

```

```

<DatePickerx:Name="datePicker" HorizontalAlignment="Left" Margin="10,54,0,0" Grid.Row="4" VerticalAlignment="Top" Width="100" Grid.ColumnSpan="2" />

```

```

<TimePickerx:Name="timePicker" HorizontalAlignment="Left" Margin="10,93,0,0" Grid.Row="4" VerticalAlignment="Top" Width="100" Grid.ColumnSpan="2" />

```

```

<Buttonx:Name="submitButton" Grid.Row="4" Content="Submit" Width="110" Click="submitButton_Click" Margin="10,131,0,202.333" />

```

```

<TextBlockHorizontalAlignment="Left" Margin="10,172,0,0" Grid.Row="4" TextWrapping="Wrap" Text="You have selected" FontSize="20" VerticalAlignment="Top" Height="47" Width="320" Grid.ColumnSpan="2" />

```

```
<TextBlockx:Name="dateTextBlock" HorizontalAlignment="Left" Margin="10,208,0,0" Grid.Row="4" TextWrapping="Wrap"
FontSize="20" VerticalAlignment="Top" Height="40" Width="320" Grid.ColumnSpan="2" />
```

```
<TextBlockx:Name="timeTextBlock" HorizontalAlignment="Left" Margin="10,253,0,0" Grid.Row="4" TextWrapping="Wrap"
FontSize="20" VerticalAlignment="Top" Height="40" Width="320" Grid.ColumnSpan="2" />
```

Listing 3

In the backend, the C# code will be like this.

```
privatevoidsubmitButton_Click(object sender, RoutedEventArgs e)
{
    //dateTextBlock.Text = datePicker.Date.Day + "/" + datePicker.Date.Month + "/" + datePicker.Date.Year;
    dateTextBlock.Text = datePicker.Date.ToString("D");
    //timeTextBlock.Text = timePicker.Time.Hours + ":" + timePicker.Time.Minutes;
    //timePicker.ClockIdentifier = Windows.Globalization.ClockIdentifiers.TwelveHour;
    timeTextBlock.Text = timePicker.Time.ToString("T");
}
```

Listing 4

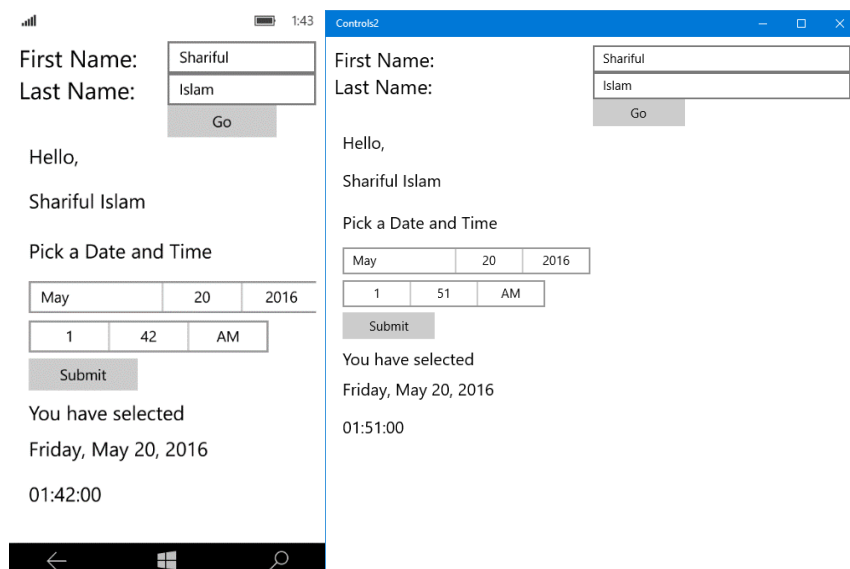
Here, in the Button event handler, we have used some methods to show date and time. Best and easy option is given here for both date and time. Others are commented out, you can try these if you want.

After, you have set all these, your design will look like this,



Figure 3

If you run the application, it will work just like this.



Summary

Hope, you can do that with me. See you with next Part 3. Till then, good bye. Have a nice day. Happy coding!

Universal Windows Platform Controls – Part 3

It's our last part of Universal Windows Platform Controls. In this chapter we will talk about Image control in Universal Windows Platform. It's really awesome and you will definitely like it. So let's learn Universal Windows Platform Image Control.

Working with Image Control

You can take an Image control from Toolbox or just write a simple code and you have to take four RadioButtons. We have talk about RadioButton in our first part of this section Universal Windows Platform Part - 1. If you are not familiar with RadioButton, feel free to take a look of it. Hence, our design looks like this picture below.

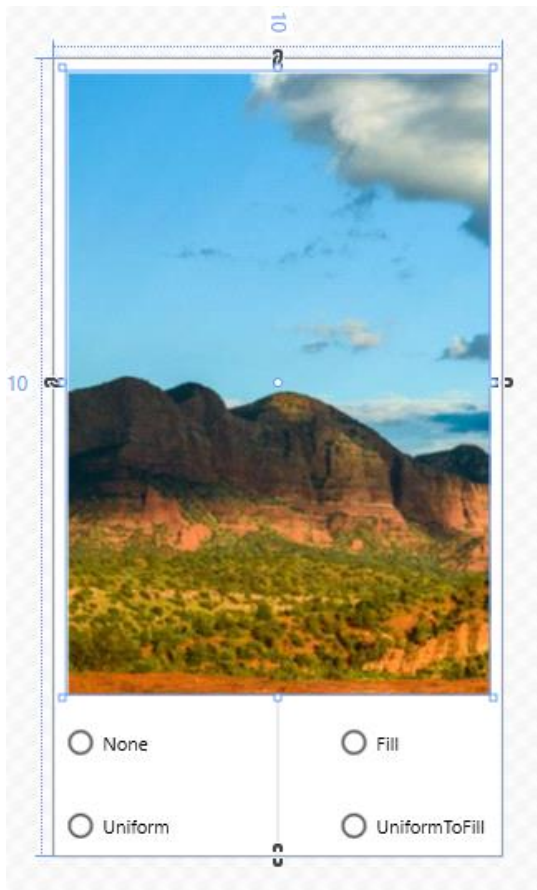


Figure 1

Adding an Image to Our Project

Now we have to do a little bit of work before proceeding. We need to add an image to our project. Just right click in the Solution Explorer and go to Add >> New Folder. Give it a name "Images".

Now, right click on the “Images” folder and go to Add >> Existing Item. Go to your destination directory to select your desired image. Select and add it.

Designing UI and Code Behind

Now, in the XAML code show the path of the image you have added in the Source property of Image control. XAML code is given below.

```
<Grid>
<Imagex:Name="Image1"
    HorizontalAlignment="Left"
    Height="473"
    VerticalAlignment="Top"
    Width="380"
    Source="Images/sample.jpg"
    Stretch="None"
    Margin="10,10,0,0"/>

<RadioButtonx:Name="NoneButton"
    Content="None"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Checked="StretchModeButton_Checked"
    Margin="10,488,0,0"/>
<RadioButtonx:Name="FillButton"
    Content="Fill"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Checked="StretchModeButton_Checked"
    Margin="222,488,0,0"/>
<RadioButtonx:Name="UniformButton" Content="Uniform"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Checked="StretchModeButton_Checked"
    Margin="10,555,0,0"/>
<RadioButtonx:Name="UniformToFillButton"
    Content="UniformToFill"
    HorizontalAlignment="Left"
    VerticalAlignment="Top"
    Checked="StretchModeButton_Checked"
    Margin="222,555,0,0"/>
</Grid>
```

Listing 1

Here, we have shown the path i.e full path of our image in line number seven. We will mainly show the four image Zooming property e.g., Fill, Uniform, Uniform to Fill and Normal (None). Our four RadioButton will handle this operation. C# code is given here.

```
privatevoidStretchModeButton_Checked(object sender, RoutedEventArgs e)
{
    RadioButton button = sender asRadioButton;
    if (Image1 != null)
    {
        switch (button.Name)
        {
            case "FillButton":
                Image1.Stretch = Windows.UI.Xaml.Media.Stretch.Fill;
                break;
            case "NoneButton":
                Image1.Stretch = Windows.UI.Xaml.Media.Stretch.None;
                break;
            case "UniformButton":
                Image1.Stretch = Windows.UI.Xaml.Media.Stretch.Uniform;
                break;
        }
    }
}
```



```

case "UniformToFillButton":
    Image1.Stretch = Windows.UI.Xaml.Media.Stretch.UniformToFill;
    break;
default:
    break;
}
}
}

```

Listing 2

Here, we have applied a very simple logic, like Switch Case operation. We just call every RadioButton by their name like in line number eight, eleven, fourteen and seventeen and call Windows Media Class. Image1 is our Image control's name. These are small lines of codes but are really helpful.

Running the Application

If you run the application, it will look exactly like this.



Figure: Windows Phone



Figure: Windows Platform.

Universal Windows Platform | Split View Control

Windows 10 brings lots of new features in Universal Windows App Platform. One of the new control that is integrated in Visual Studio is Split View control. This control is the one that helps in navigation better. It gives a whole new experience to navigate between the pages. Moreover, Split View is not only for navigation but you can also customize it as per your application needs.

The Split View Control can be used to make a nav pane pattern. To build this pattern, add an expand/collapse button (the "hamburger" button) and a list view is required for the split view control.

Examples

The Split View Control in its default form is a basic container. With a button and a list view added, the Split View Control is ready as a navigation menu. Here are the examples of the Split View as a navigation menu in expanded and compact modes.

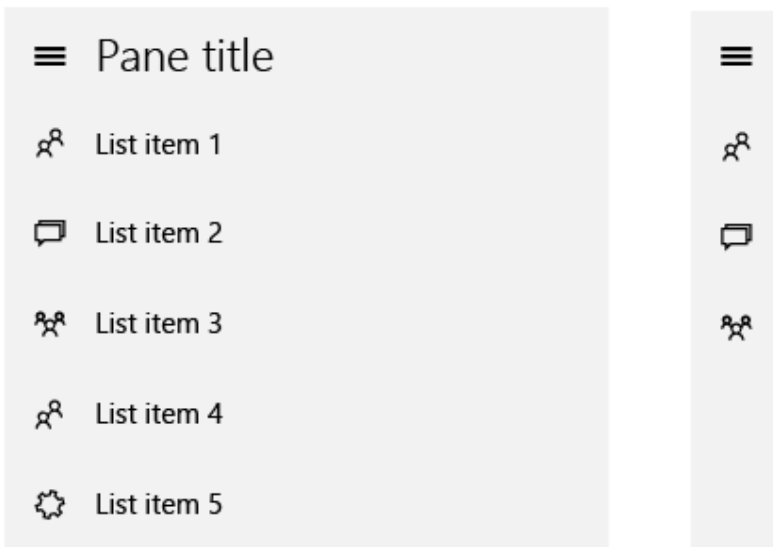


Figure: 1

Creating a Project with Split View Control

First of all, create a new project and in MainPage.xaml inside, the Grid makes a new SplitView control.

The code will be as follows-

```
<SplitViewx:Name="SplitView" OpenPaneLength="240" CompactPaneLength="48" DisplayMode="CompactOverlay"
IsPaneOpen="False" PaneBackground="DarkGray">
    <SplitView.Pane>
        <StackPanelx:Name="SplitViewPanePanel">
            <RadioButtonx:Name="BackRadioButton" Click="BackRadioButton_Click"
                Style="{StaticResourceNavRadioButtonStyle}" Tag="玊" Background="Gray" Content="Back"
                GroupName="Back" />
            <RadioButtonx:Name="HamburgerRadioButton" Click="HamburgerRadioButton_Click"
                Style="{StaticResourceNavRadioButtonStyle}" Tag="璠" Content="Menu"
                GroupName="Hamburger" />
            <RadioButtonx:Name="HomeRadioButton" Click="HomeRadioButton_Click"
                Style="{StaticResourceNavRadioButtonStyle}" Tag="璡" Content="Home"
                GroupName="Navigation" />
            <RadioButtonx:Name="SettingsRadioButton" Click="SettingsRadioButton_Click"
                Style="{StaticResourceNavRadioButtonStyle}" Tag="玊" Content="Settings"
                GroupName="Navigation" />
        </StackPanel>
    </SplitView.Pane>
    <SplitView.Content>
        <Frame Name="MyFrame">
            </Frame>
        </SplitView.Content>
</SplitView>
```

Listing 1

In SplitView control, there are two sections, one is “SplitView.Pane” and another is “SplitView.Content”. “SplitView.Pane” holds all the list items of your navigation links, which you can see in Figure 1 below and “SplitView.Content” is the main frame of your application, which you can populate with other controls and application data whatsoever. We have taken a Frame inside the “SplitView.Content” because we want to make the Split View fixed and load different pages as required. Therefore, the user will notice that a new frame will be loaded instead of a new page. That will give a great experience to your application.

Simple custom RadioButtonStyle

To set the icon of the “SplitView.Pane”, we have used a custom style of Radio Button. In the Radio Button, there is a Tag property which you can use to make use of the Character Map Icons.

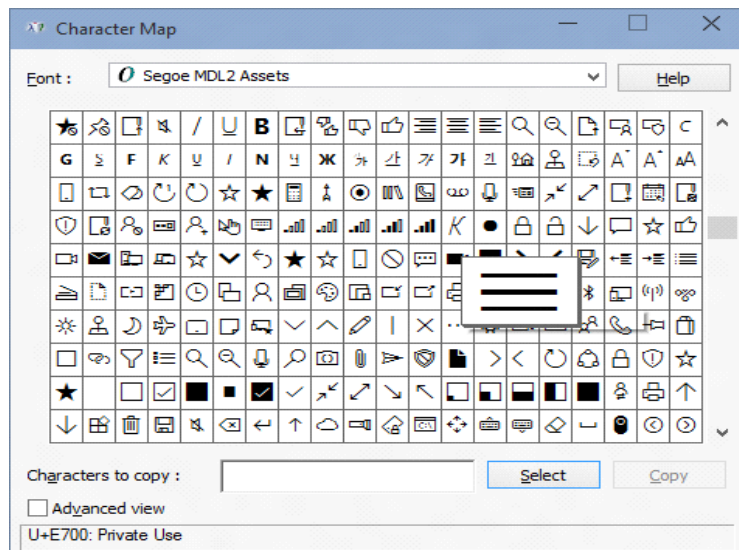


Figure: 2

We have put the style code outside the grid. The custom style is given below.

```
<Page.Resources>
  <ResourceDictionary>
    <SolidColorBrushx:Key="NavButtonPressedBackgroundBrush" Color="White" Opacity="0.3" />
    <SolidColorBrushx:Key="NavButtonCheckedBackgroundBrush" Color="White" Opacity="0.2" />
    <SolidColorBrushx:Key="NavButtonHoverBackgroundBrush" Color="White" Opacity="0.1" />

    <Style x:Key="NavRadioButtonStyle" TargetType="RadioButton">
      <Setter Property="Background" Value="Transparent"/>
      <Setter Property="Padding" Value="3"/>
      <Setter Property="HorizontalAlignment" Value="Stretch"/>
      <Setter Property="VerticalAlignment" Value="Center"/>
      <Setter Property="HorizontalContentAlignment" Value="Left"/>
      <Setter Property="VerticalContentAlignment" Value="Center"/>
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplateTargetType="RadioButton">
            <BorderBorderBrush="{TemplateBindingBorderBrush}" BorderThickness="{TemplateBindingBorderThickness}" Background="{TemplateBindingBackground}">
              <VisualStateManager.VisualStateGroups>
                <VisualStateGroupx:Name="CommonStates">
                  <VisualStatex:Name="Normal"/>
                  <VisualStatex:Name="PointerOver">
                    <Storyboard>
                      <ObjectAnimationUsingKeyFramesStoryboard.TargetProperty="Background"Storyboard.TargetName="BackgroundGrid">
                        <DiscreteObjectKeyFrameKeyTime="0" Value="{StaticResourceNavButtonHoverBackgroundBrush}" />
                      </ObjectAnimationUsingKeyFrames>
                    </Storyboard>
                  </VisualState>
                  <VisualStatex:Name="Pressed">
```

```

        <Storyboard>
            <ObjectAnimationUsingKeyFramesStoryboard.TargetProperty="Background"Storyboard.TargetName="BackgroundGrid">
                <DiscreteObjectKeyFrameKeyTime="0"
                    Value="{StaticResourceNavButtonPressedBackgroundBrush}"/>
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualStateName="Disabled" />
</VisualStateGroup>
<VisualStateGroupName="CheckStates">
    <VisualStateName="Checked">
        <Storyboard>
            <ObjectAnimationUsingKeyFramesStoryboard.TargetProperty="Background"Storyboard.TargetName="BackgroundGrid">
                <DiscreteObjectKeyFrameKeyTime="0"
                    Value="{StaticResourceNavButtonCheckedBackgroundBrush}"/>
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualStateName="Unchecked"/>
    <VisualStateName="Indeterminate"/>
</VisualStateGroup>
<VisualStateGroupName="FocusStates">
    <VisualStateName="Focused"/>
    <VisualStateName="Unfocused"/>
    <VisualStateName="PointerFocused"/>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid Name="BackgroundGrid" Background="Transparent"VerticalAlignment="Stretch">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="48"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <TextBlockFontSize="34" Height="38" Text="{TemplateBinding Tag}"FontFamily="Segoe MDL2 Assets" Margin="5,8,5,5"VerticalAlignment="Center"HorizontalAlignment="Center"/>
    <ContentPresenterName="ContentPresenter"AutomationProperties.AccessibilityView="Raw"ContentTemplate="{TemplateBindingContentTemplate}"ContentTransitions="{TemplateBindingContentTransitions}" Content="{TemplateBinding Content}"Grid.Column="1"HorizontalAlignment="{TemplateBindingHorizontalContentAlignment}" Margin="{TemplateBinding Padding}"TextWrapping="Wrap"VerticalAlignment="{TemplateBindingVerticalContentAlignment}"/>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>
</Page.Resources>

```

Listing: 2

You can also create your custom styles using Blend. To do this, open a blank project in Blend or open your existing project like below.

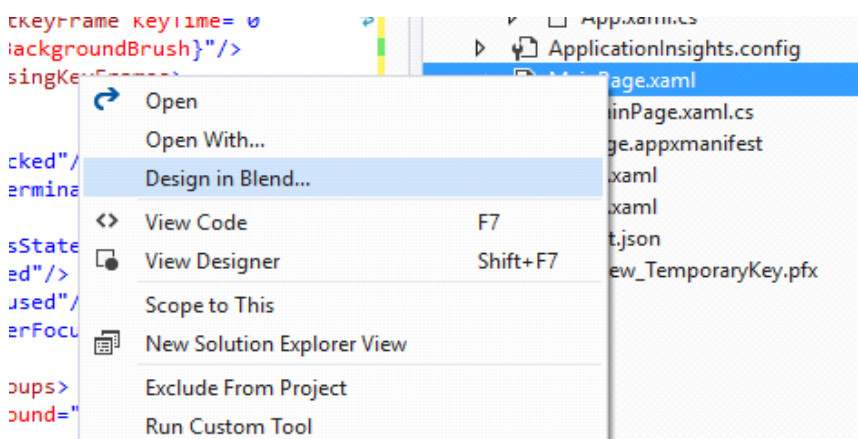


Figure: 3

In main Grid, take a Radio Button control.

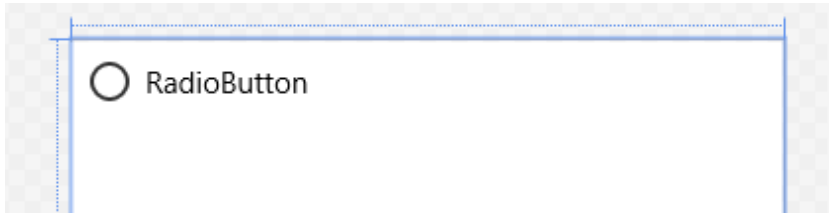


Figure: 4

Right click on the control and select Edit Template and under Edit Template option select Edit a Copy.

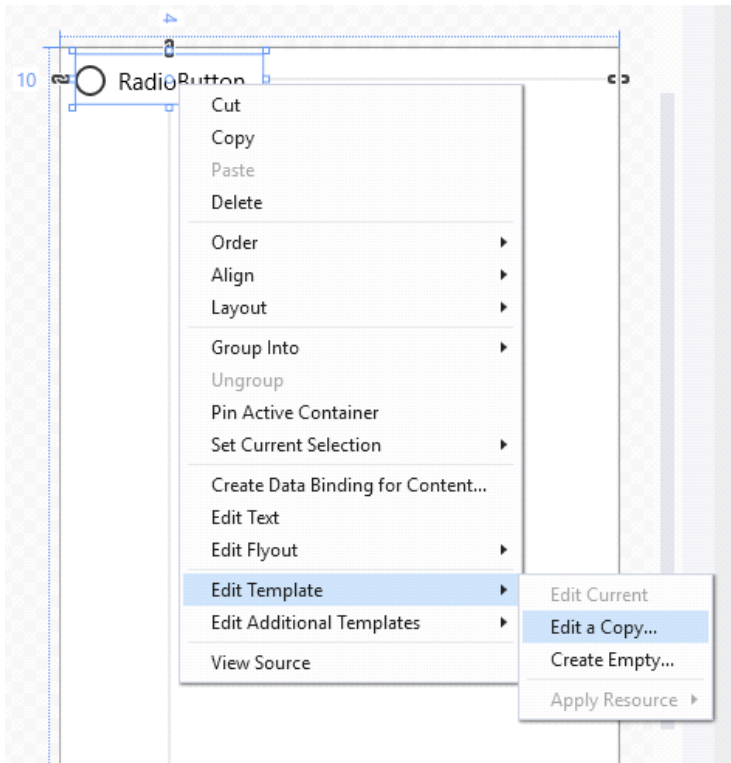


Figure: 5

Then you can customize the style according to your requirements.

After everything has been set, the MainPage.xaml will look like this.

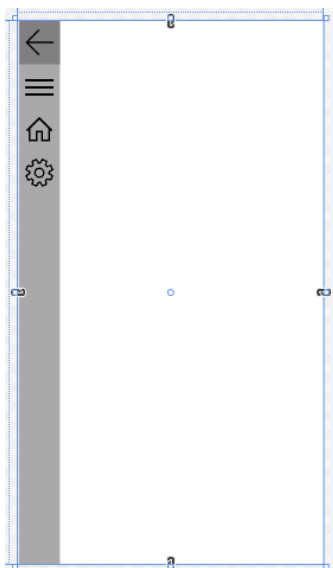


Figure: 6

Adding Extra Pages

Now, create two blank pages named Page1.xaml and Page2.xaml. In Page1.xaml, change the code as shown below.

```
<Grid Background="LightBlue">
    <TextBlockFontSize="36" Text="Home Page" Margin="12,17,0,17"/>
</Grid>
```

Listing: 3

Similarly, in Page2.xaml change the code as shown below.

```
<Grid Background="SteelBlue">
    <TextBlockFontSize="36" Text="Settings Page" Margin="12,17,0,17"/>
</Grid>
```

Listing: 4

Handle events in code-behind MainPage.xaml.cs

The event handlers of the Split View's Radio Buttons are as follows.

```
public MainPage()
{
    this.InitializeComponent();
    MyFrame.Navigate(typeof(Page1));
}

private void BackRadioButton_Click(object sender, RoutedEventArgs e)
{
    if (MyFrame.CanGoBack)
    {
        MyFrame.GoBack();
    }
}

private void HamburgerRadioButton_Click(object sender, RoutedEventArgs e)
{
    if (!this.SplitView.IsPaneOpen)
    {
        this.SplitView.IsPaneOpen = true;
    }
    else
    {
        this.SplitView.IsPaneOpen = false;
    }
}

private void HomeRadioButton_Click(object sender, RoutedEventArgs e)
{
    MyFrame.Navigate(typeof(Page1));
}

private void SettingsRadioButton_Click(object sender, RoutedEventArgs e)
{
    MyFrame.Navigate(typeof(Page2));
}
```

Listing: 5

Here, we have loaded the Page1 inside the Frame name "MyFrame" in MainPage.xaml. In MainPage.xaml.cs, we have navigated to Page1 in Constructor. Therefore, when the application starts, it will open the Page1 under MyFrame control. Back Button event handler checks whether the frame

can go back or not. Hamburger Button checks if the SplitView is open or not. If not open, it will open it and if it is open, then it will close it. Home and Settings Button do the basic navigation service.

Running the Application

If you run the application in local machine, it will look like this.

Local Machine:

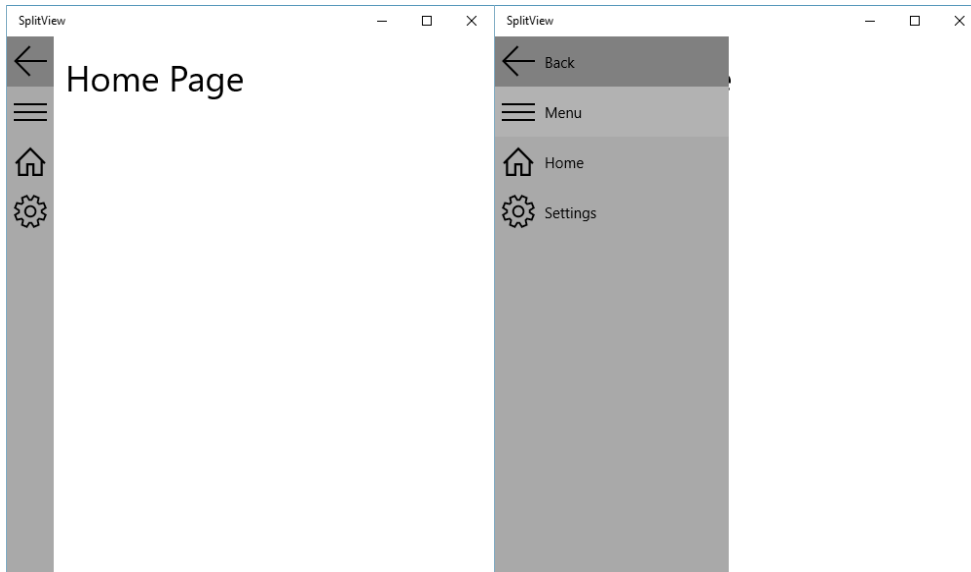


Figure: 7

If you click the Settings button, it will navigate to the Settings page.

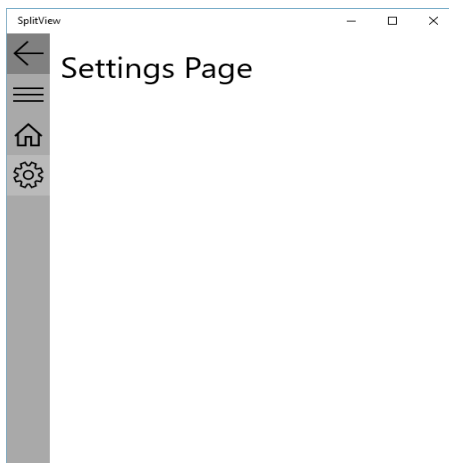


Figure: 8

Running in the Windows Phone will look like the one shown below:

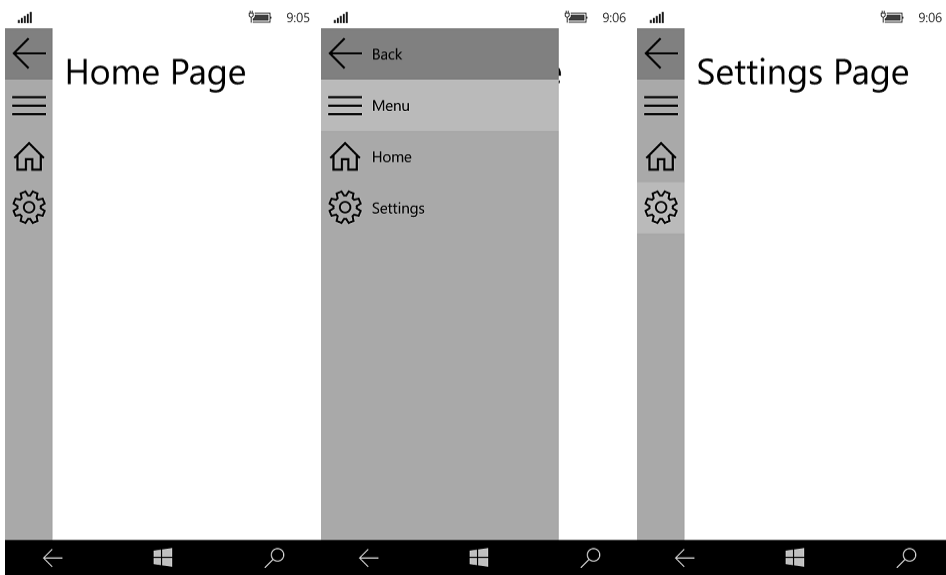


Figure: 9

Improve it a bit for Windows Phone

Sometimes your design may not fit for both desktop and mobile devices. We have implemented a Back Button in our Split View control, as desktop application does not have an integrated Back Button but mobile device has its own hardware or OS (Operating System) which has an integrated Back Button. By determining device type, we can customize our application UI (User Interface). You can do this, by simply installing a NuGet package named “WindowsStateTriggers”.

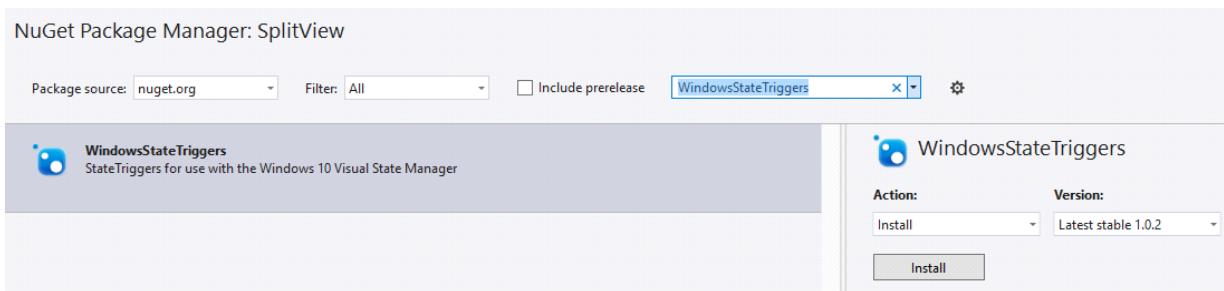


Figure: 10

Install it and now put the reference in your MainPage.xaml, inside the Page tag.

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

  xmlns:triggers="using:WindowsStateTriggers">
```

Listing: 6

We have named the reference “triggers”. Now, we can use it in our XAML code. To determine the device type, we have to use State Trigger like the one shown below.

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup>
    <VisualStateEx:Name="phone">
      <VisualState.StateTriggers>
        <triggers:DeviceFamilyStateTriggerDeviceFamily="Mobile" />
      </VisualState.StateTriggers>
    </VisualState.Setters>
```

```

        <Setter Target="BackRadioButton.Visibility" Value="Collapsed" />
    </VisualState.Setters>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>

```

Listing: 7

In State Triggers, we have set Device Family to “Mobile” and set the value of Back Button to Collapsed. We have to put this inside the main Grid. This will hide the Back Button when the application will run in mobile devices. If we run the application, it will look like this.

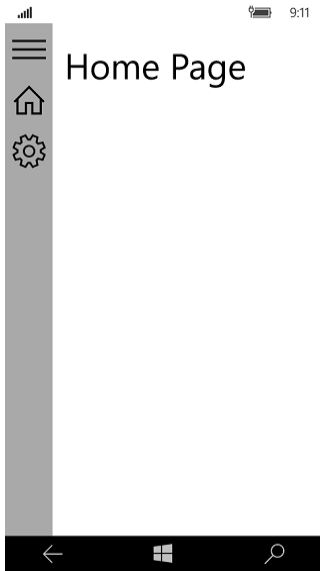


Figure: 10

You can learn further about Windows State Triggers at <https://github.com/dotMorten/WindowsStateTriggers>

Important things to notice

In SplitView control, several things are required to be considered. First of all, there is a Display Mode. There are four types of Display Mode, Compact Inline, Compact Overlay, Inline and Overlay. If you use Compact Inline, the Split View Content will move right unlike using Compact Overlay.

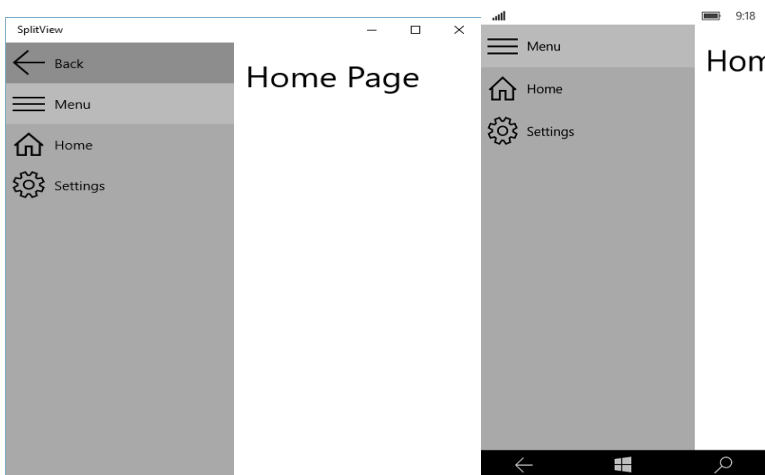


Figure: 11

OpenPaneLength determines the width of the SplitView Pane’s width and compactPaneLength determines the iconic view’s width. You can also set PaneBackgroundColor. We have set here Dark Gray. Moreover, the Tag property of Radio Button is to set the icon of the Radio Button. You can

choose it from Character Map, shown in Figure 2 above. RadioButtons have a Tag set to a certain character. The new character set is Segoe MDL2 Assets and the characters used are from that set.

Split View is a very useful control that was introduced in Windows 10 Universal App. It will give you the power to do whatever you want. You can use it instead of Bottom App Bar but it can still be used alongside. Hope you got the understanding of using Split View control and can get started using in your application. Happy coding!

Universal Windows Platform | List Box

One of the common and important controls of Windows 10 Universal App is List Box control. It is really helpful to make a long list of elements. You can populate it with some data, pictures or anything that you want to show in your application. List Box can be implemented easily using XAML. In this example, we are going to use our previous project SplitView control and make the Split View Pane items with the help of List Box. So, let's get started.

Creating a Project with Split View Control

First of all, create a new project and in MainPage.xaml inside the Grid to make a new SplitView control.

```
<SplitViewx:Name="SplitView"OpenPaneLength="200"CompactPaneLength="50"
    DisplayMode="CompactOverlay"IsPaneOpen="False"PaneBackground="DarkGray">
    <SplitView.Pane>
        <!--Pane Items-->
    </SplitView.Pane>
    <SplitView.Content>
        <!--Content View-->
    </SplitView.Content>
</SplitView>
```

Listing: 1

Previously, we have created the Pane items with Radio Buttons with custom styles. Now we will work with ListBox items and the work will be much easier for us.

Creating ListBox Items

ListBox is a very simple control with inline elements of ListBox items. The basic structure of ListBox is given below.

```
<ListBox>
    <ListBoxItem Content="First" />
    <ListBoxItem Content="Second" />
    <ListBoxItem Content="Third" />
</ListBox>
```

Listing: 2

We can declare as many items inside the ListBox control. Now, replace the commented Pane Items with the code below.

```
<ListBoxSelectionMode="Single" SelectionChanged="ListBox_SelectionChanged">
    <ListBoxItem Name="Back" Background="Gray">
        <StackPanel Orientation="Horizontal">
            <TextBlockFontFamily="Segoe MDL2 Assets"FontSize="30" Text="□" />
            <TextBlockFontSize="24" Margin="10,0,0,0">Back</TextBlock>
        </StackPanel>
    </ListBoxItem>
```

```

<ListBoxItem Name="Hamburger">
    <StackPanel Orientation="Horizontal">
        <TextBlockFontFamily="Segoe MDL2 Assets"FontSize="30" Text="☰" />
        <TextBlockFontSize="24" Margin="10,0,0,0">Menu</TextBlock>
    </StackPanel>
</ListBoxItem>
<ListBoxItem Name="Home">
    <StackPanel Orientation="Horizontal">
        <TextBlockFontFamily="Segoe MDL2 Assets"FontSize="30" Text="☑" />
        <TextBlockFontSize="24" Margin="10,0,0,0">Home</TextBlock>
    </StackPanel>
</ListBoxItem>
<ListBoxItem Name="Settings">
    <StackPanel Orientation="Horizontal">
        <TextBlockFontFamily="Segoe MDL2 Assets"FontSize="30" Text="⚙" />
        <TextBlockFontSize="24" Margin="10,0,0,0">Settings</TextBlock>
    </StackPanel>
</ListBoxItem>
</ListBox>

```

Listing: 3

Here we took a ListBox and there are four List Box items like previously we had four Radio Buttons. Each item represents correspondent SplitView Pane item. Inside the ListBoxItem, there is a StackPanel to hold the icon and the text of the items. StackPanel arranges the TextBlocks in a same alignment. In first TextBlock, Font Family is Sagoe MDL2 Assets, which is the same as Character Map special character set and Text is just the following character.

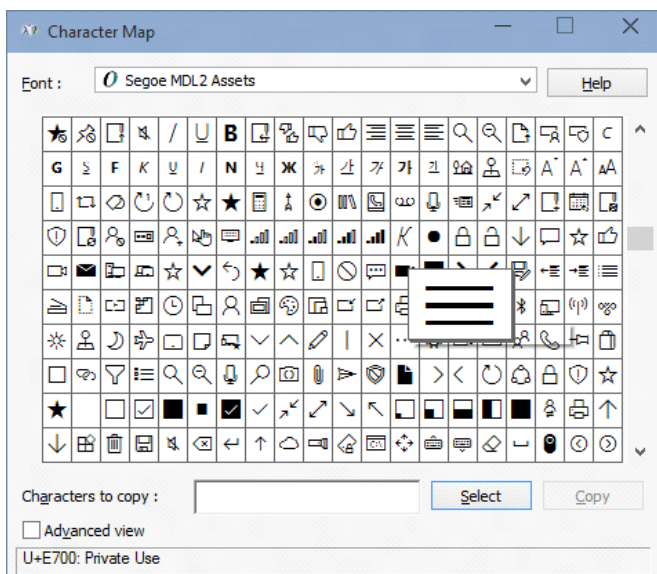


Figure: 1

Adding Extra Pages

Now, create two blank pages named Page1.xaml and Page2.xaml. In Page1.xaml, change the code as shown below.

```

<Grid Background="{ThemeResourceApplicationPageBackgroundThemeBrush}">
    <TextBlockFontSize="36" Text="Home Page" Margin="12,17,0,17"/>
</Grid>

```

Listing: 4

Similarly, in Page2.xaml change the code like below.

```

<Grid Background="{ThemeResourceApplicationPageBackgroundThemeBrush}">
    <TextBlockFontSize="36" Text="Settings Page" Margin="12,17,0,17"/>
</Grid>

```

Listing: 5

Handle events in code-behind MainPage.xaml.cs

The event handlers of the Split View's Radio Buttons are as follows.

```
private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (Back.IsSelected)
    {
        if (MyFrame.CanGoBack)
        {
            MyFrame.GoBack();
        }
    }
    elseif (Hamburger.IsSelected)
    {
        if (!this.SplitView.IsPaneOpen)
        {
            this.SplitView.IsPaneOpen = true;
        }
        else
        {
            this.SplitView.IsPaneOpen = false;
        }
    }
    elseif (Home.IsSelected)
    {
        MyFrame.Navigate(typeof(Page1));
    }
    elseif (Settings.IsSelected)
    {
        MyFrame.Navigate(typeof(Page2));
    }
    else
    {
        // do nothing
    }
}
```

Listing: 6

If you notice, we do not have any extra Button control here and it is just the SelectionChanged event of the ListBox control. Every ListBoxItem has a unique name so we check whether it is selected or not. You can also see the previous article at <http://bit.ly/1O626HP>.

Running the Application

If you run the application, it will look like this.

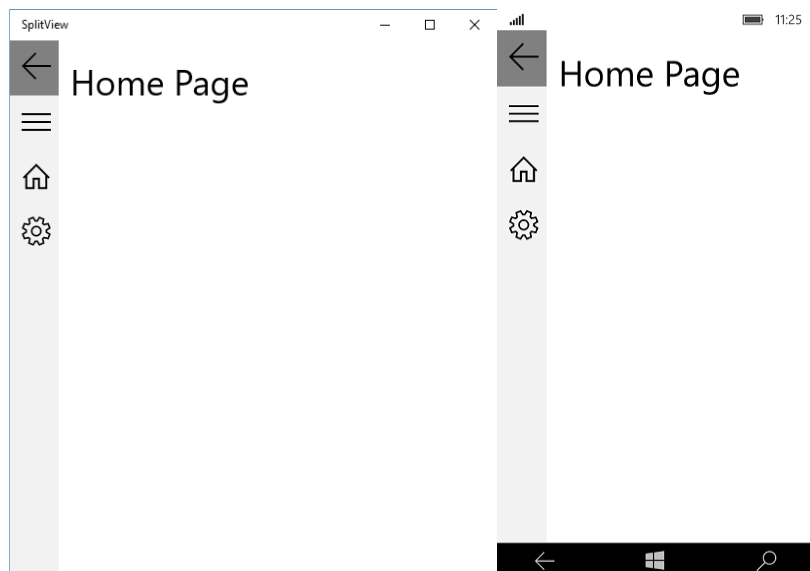


Figure: 2

ListBox is a common and a useful control. You can use it for static data as well as some other data source. Hope this helps you to understand the basic implementation of ListBox in Universal Windows Application. Happy Coding!

Universal Windows Platform | XAML Styling

In this chapter, we'll talk about XAML styling. How can you make your XAML controls more beautiful and customized? If you search "Universal Windows Platform XAML style", you'll get some helpful references. Styling XAML is not only for customizing your controls but it also makes code cleaner and easily readable. So let's crack in Windows Phone XAML Styling.

We will just try to explain on how you can use XAML styling in you existent projects. I am just going to modify my existing user control to show you the procedure. If you have read my [Windows Phone Controls HYPERLINK "](#) article, then you can understand the difference between the previous and current XAML code. I will not modify all the controls but the "Popup Window". I have used a "User Control" and will just modify that page.

Creating a New Project and Add a User Control

First of all, take a new Project and add a new "User Control". We have used this XAML code in our previous Project.

```
<Grid>
  <BorderBorderBrush="{StaticResourceApplicationForegroundThemeBrush}"BorderThickness="1"
  Background="{StaticResourceApplicationPageBackgroundThemeBrush}">
    <BorderBorderBrush="{StaticResourceApplicationForegroundThemeBrush}"BorderThickness="1">
      <BorderBorderBrush="{StaticResourceApplicationForegroundThemeBrush}"BorderThickness="1">
        <BorderBorderBrush="{StaticResourceApplicationForegroundThemeBrush}"BorderThick
        ness="1">
          <StackPanel Orientation="Vertical" Height="200"
          Width="200"VerticalAlignment="Center">
            <TextBlock Text="This is a
            Popup!"VerticalAlignment="Center"HorizontalAlignment="Center"
            Margin="0,60,0,0"/>
            <TextBlock Text="Hit the button again to hide
            me"VerticalAlignment="Center"HorizontalAlignment="Center"
            Margin="0,10,0,0"TextWrapping="Wrap"TextAlignment="Center"/>
            <ButtonHorizontalAlignment="Center" Content="Close Popup"
            Click="ClosePopup" />
          </StackPanel>
        </Border>
      </Border>
    </Border>
  </Grid>
```

Listing 1

The design will look like this.



Figure 1

Now, we will use XAML styling in the same XAML code and make it clean and customize. To do so, you have to use resources as shown below.

```
<UserControl
    ...
    d:DesignWidth="400">
    <UserControl.Resources>
        ...
    </UserControl.Resources>
    <Grid>
        ...
    </Grid>
</UserControl>
```

Listing 2

Creating Styles

All you’ve to do, is put all your style properties inside the Resources tag. First of all, we will create a “Border Style” for our “Border” control.

```
<UserControl.Resources>
    <Style:Key="BorderStyle"TargetType="Border">
        <Setter Property="BorderThickness" Value="2"/>
        <Setter Property="CornerRadius" Value="0,10,0,10"/>
    </Style>
</UserControl.Resources>
```

Listing 3

Note: If you are using this in Blank of Basic pages, the code will be like this.

```
<Page.Resources>
    <Style:Key="BorderStyle"TargetType="Border">
        <Setter Property="BorderThickness" Value="2"/>
        <Setter Property="CornerRadius" Value="0,10,0,10"/>
    </Style>
</Page.Resources>
```

Listing 4

As we are using a “User Control”, so we used “UserControl.Resources”.

Here, we are considering only one “Border” control. If you look above, the code we gave is the style name “BorderStyle” and set target to “Border”. In the control that you will work, you have to give a unique name and set a target of that control. Also, we have set a property name “BorderThickness” and set value to “2”, which will make the thickness of the border’s outer edges. We have also set

“CornerRadius” to “0,10,0,10”, which will make the upper right and lower left corner edges little bit round.

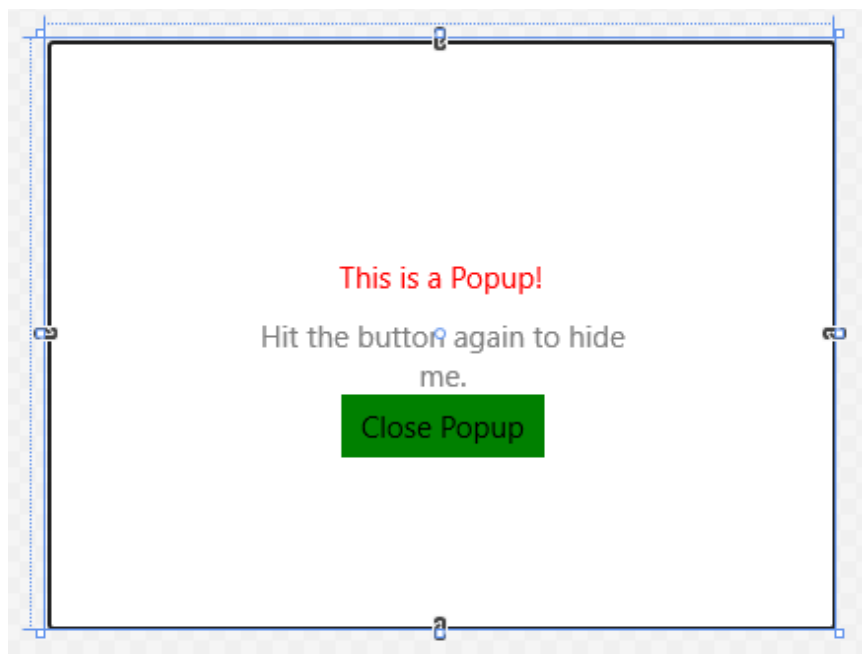


Figure 2

Now, similarly we have added “TextBox” and “Button” styles.

```
<UserControl.Resources>
  <Stylex:Key="BorderStyle"TargetType="Border">
    <Setter Property="BorderThickness" Value="2"/>
    <Setter Property="CornerRadius" Value="0,10,0,10"/>
  </Style>
  <Stylex:Key="StackPanelStyle"TargetType="StackPanel">
    <Setter Property="Orientation" Value="Vertical"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Height" Value="200"/>
    <Setter Property="Width" Value="200"/>
  </Style>
  <Stylex:Key="ButtonStyle"TargetType="Button">
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="Content" Value="Close Popup"/>
    <Setter Property="Background" Value="Green"/>
  </Style>
  <Stylex:Key="TextBlockStyle"TargetType="TextBlock">
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="Text" Value="This is a Popup!"/>
    <Setter Property="Margin" Value="0,60,0,0"/>
    <Setter Property="Foreground" Value="Red"/>
  </Style>
  <Stylex:Key="TextBlockStyle1"TargetType="TextBlock">
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="TextAlignment" Value="Center"/>
    <Setter Property="TextWrapping" Value="Wrap"/>
    <Setter Property="Text" Value="Hit the button again to hide me."/>
    <Setter Property="Margin" Value="0,10,0,0"/>
    <Setter Property="Foreground" Value="Gray"/>
  </Style>
</UserControl.Resources>
```

Listing 5

If you look at the old XAML code, you can see all the properties are here exactly exception in “Button” click event. You have to put this event in the main Grid’s “Button” control code.


```

<Grid>
  <Border BorderBrush="{StaticResource ApplicationForegroundThemeBrush}"
    Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
    Style="{StaticResource BorderStyle}">
    <StackPanel Style="{StaticResource StackPanelStyle}">
      <TextBlock Style="{StaticResource TextBlockStyle}" />
      <TextBlock Style="{StaticResource TextBlockStyle1}" />
      <Button Style="{StaticResource ButtonStyle}" Click="ClosePopup" />
    </StackPanel>
  </Border>
</Grid>

```

Listing 6

All you need to do is, just reference the styles to corresponding controls. Like in “Border” control, we have used “Style={StaticResourceBorderStyle}”. After the “StaticResource”, name the Style name.

Put Your Styles Separate

Another important thing you can do is to separate the XAML styling into a different location. To make XAML clean, just open “App.xaml” and put the same code there, like this

```

<Application
...
  >
  <!--Application Resources-->
  <Application.Resources>
    <Style:Key="BorderStyle" TargetType="Border">
      <Setter Property="BorderThickness" Value="2"/>
      <Setter Property="CornerRadius" Value="5"/>
    </Style>
    <Style:Key="StackPanelStyle" TargetType="StackPanel">
      <Setter Property="Orientation" Value="Vertical"/>
      <Setter Property="VerticalAlignment" Value="Center"/>
      <Setter Property="Height" Value="200"/>
      <Setter Property="Width" Value="200"/>
    </Style>
  </Application.Resources>
</Application>

```

Listing 7

Only difference is the tag, here it should be “Application.Resources”, because it’s in “App.xaml” file. Therefore, the tag structure should be like “Type.Resources”. Here type can “Page”, “Application”, “UserControl” etc.

Now, in “Main Page.xaml” take Button control to show the “Popup Window”.

```

<Page.Resources>
  <Style:Key="ButtonStyle" TargetType="Button">
    <Setter Property="Name" Value="PopupButton"/>
    <Setter Property="HorizontalAlignment" Value="Left"/>
    <Setter Property="VerticalAlignment" Value="Top"/>
    <Setter Property="Width" Value="140"/>
    <Setter Property="Margin" Value="10,0,0,0"/>
    <Setter Property="Content" Value="Show Popup"/>
  </Style>
</Page.Resources>

<Grid>
  <Button Style="{StaticResource ButtonStyle}" Click="PopupButton_Click"/>
</Grid>

```

Listing 8

Code Behind

C# code of this “Button” event handler is given below.

```
private void PopupButton_Click(object sender, RoutedEventArgs e)
{
    if (!popup.IsOpen)
    {
        popup.Child = new PopupPanel();
        popup.VerticalOffset = 200.0;
        popup.HorizontalOffset = 160.0;
        popup.IsOpen = true;
    }
}
```

Popup popup = new Popup();

Listing 9

Running the Application

If you run the application, it will look like this.

Windows Phone Emulator:

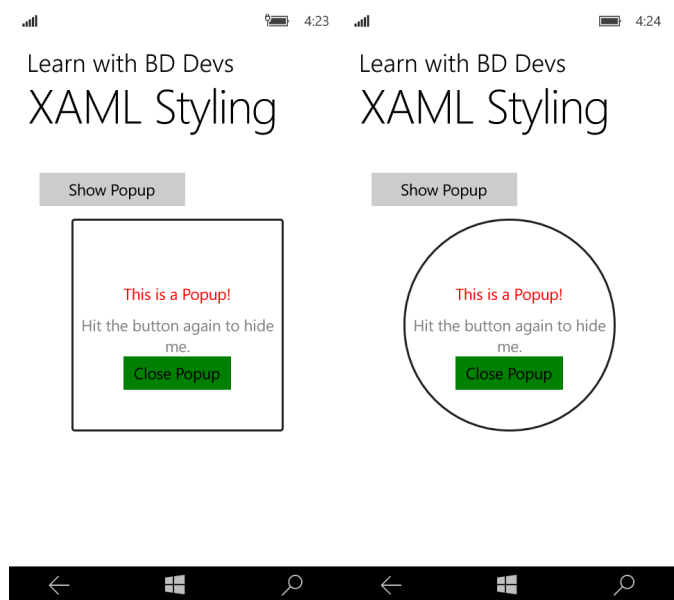


Figure 3

Figure 4

Windows 10 Local Machine:

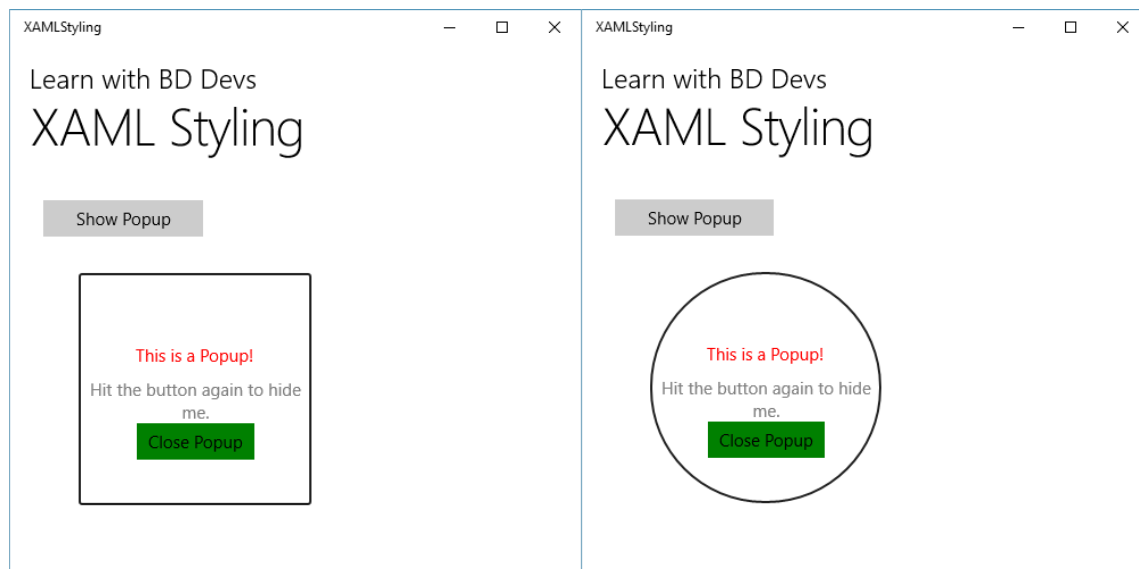


Figure 5

Figure 6

Here, we have another sample, where “Popup Window” is round. You can simply do that by just changing this code in “PopupPanel.xaml”

```

<Page.Resources>
  <Style:Key="BorderStyle"TargetType="Border">
    <Setter Property="BorderThickness" Value="2"/>
    <Setter Property="CornerRadius" Value="100"/>
  </Style>
</Page.Resources>

```

Listing 10

Universal Windows Platform | Package.appxmanifest

In this chapter, we’ll talk about Windows Phone “Package.appxmanifest”. It’s really very important to make your app fully complete because just designing and lots of code does not make a good app and the looks of the app must be very attractive. Look and feel is very important to attract a customer. Moreover, not only looks but also the capabilities and dependencies of an app needs to be set in “Package.appxmanifest”.

Exploring Package.appxmanifest

So let’s explore the Windows Phone “Package.appxmanifest”. Just open up the “Package.appxmanifest” from your “Solution Explorer” and you can see the similar picture below.

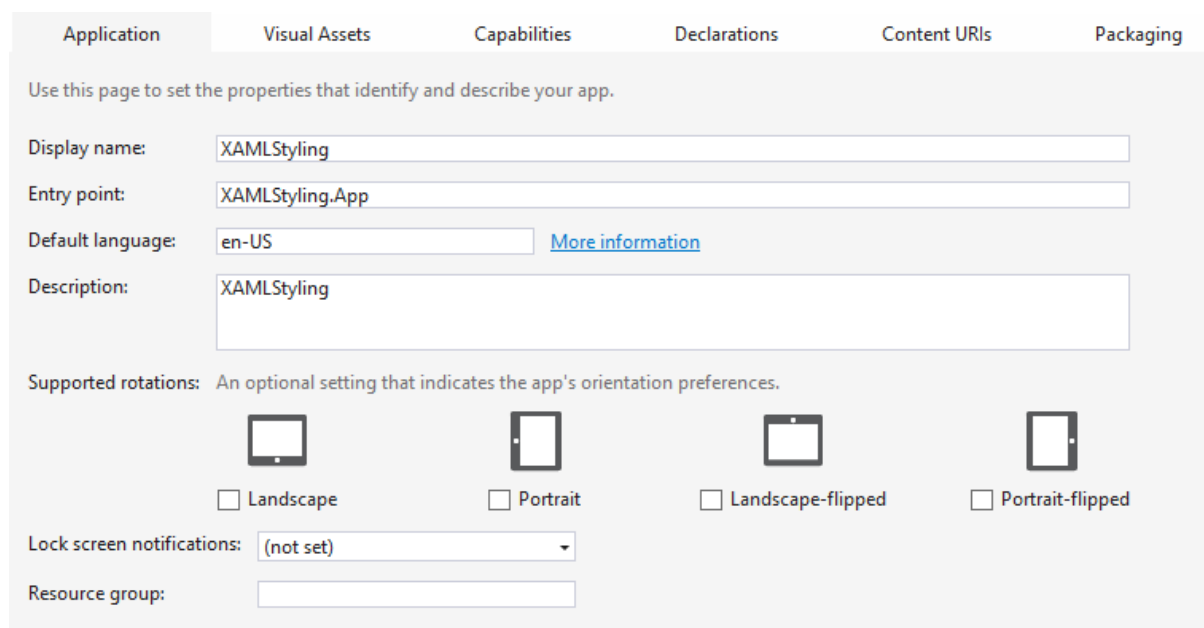


Figure 1

I have used my previous example of [Windows Phone XAML Styling](#) here. You can use any of the existing application or just create a new one.

Changing Application Title

If you take a look at the picture, “Display name” and “Description” have that same content “XAMLStyling”. Yes, it’s our app name, we have given before but does it look good, when we run the application?

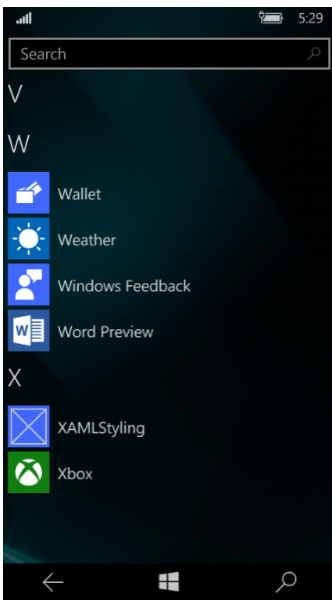


Figure 2.1

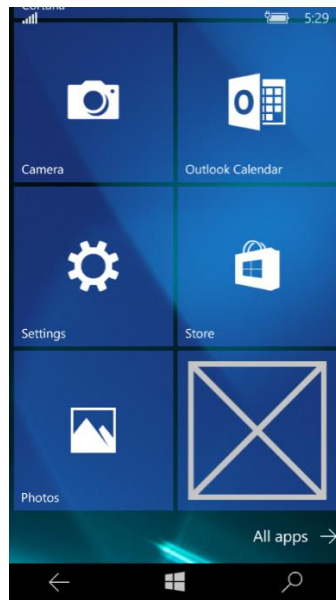


Figure 2.2

The answer will obviously be “No”. So we just need to do little bit of work. Just give a space between “XAML” and “Style”, so it will be “XAML Style”.

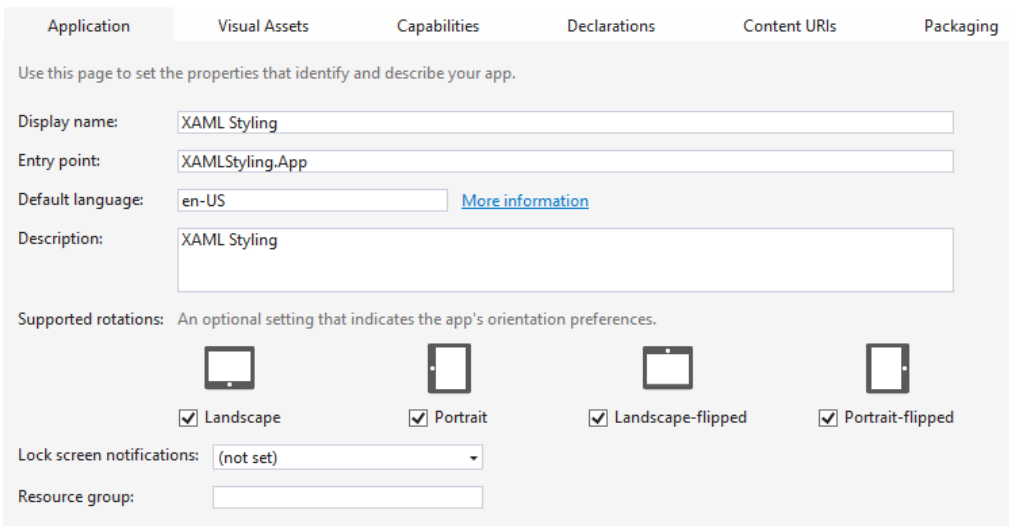


Figure 3

One more thing we’ve to do is, switch to the “Visual Assets” tab and check the “Check Box” below “Title”.

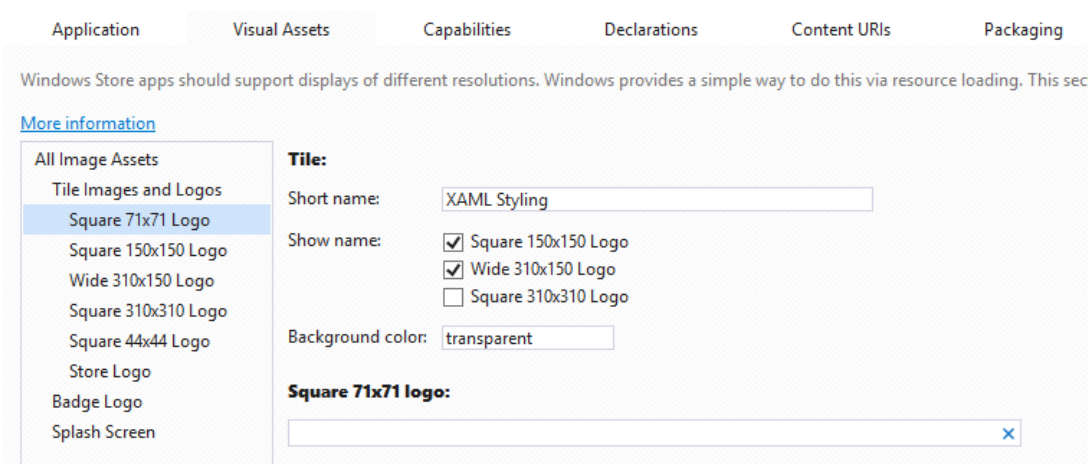


Figure 4

After that, you'll get some good visualization of your app's tiles.

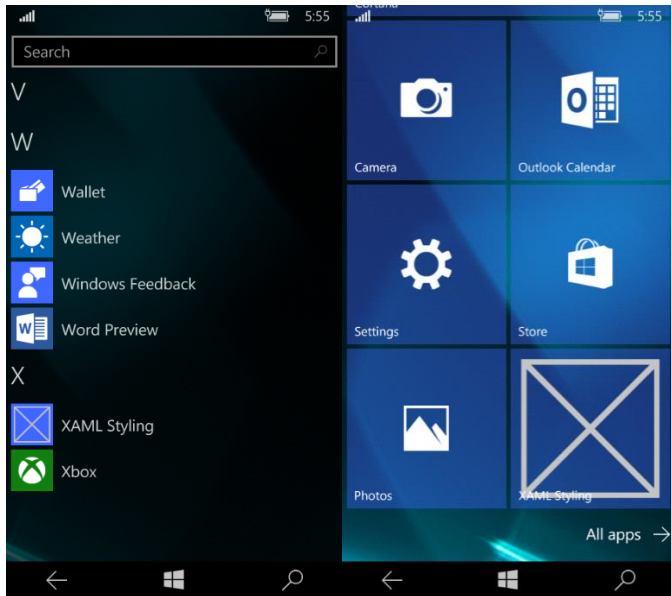


Figure 5.1

Figure 5.2

Changing Application Logo

Very important thing is to change the application logo. It's your app's unique identity. So, select the list item one by one and change your logo but before that, you have to upload your logo to your project solution. To do that, right click on the project and add a new folder. Give it a name "Images". Right click on the folder and click Add >> Existing Item.

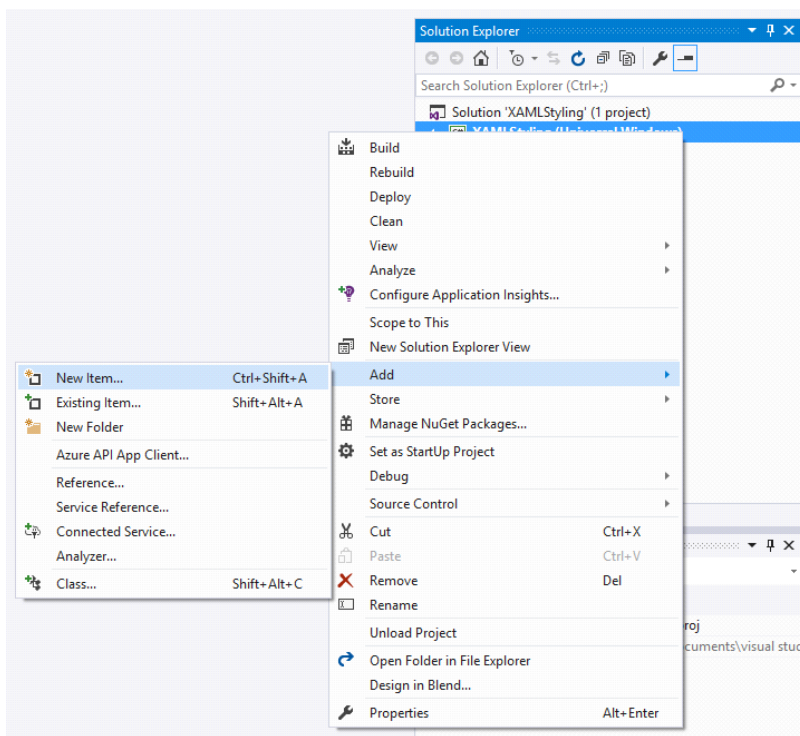


Figure 6

Upload all the required logos of corresponding resolution.

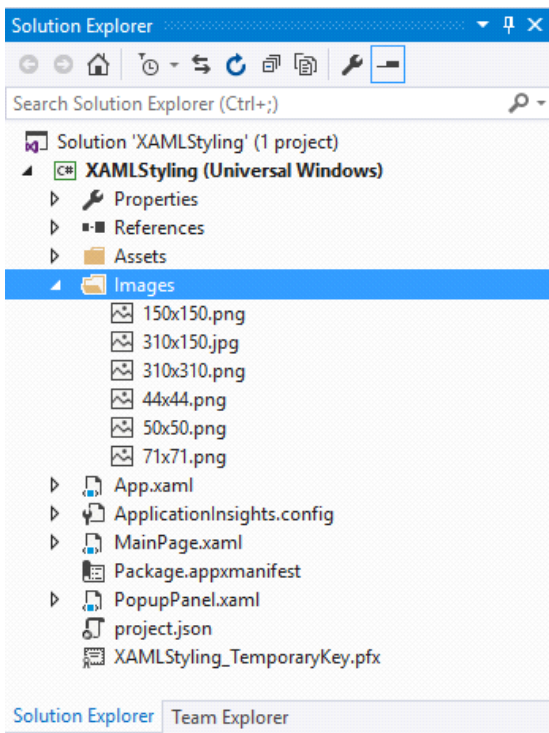


Figure 7

Now, locate all the logos in “Visual Assets” section.

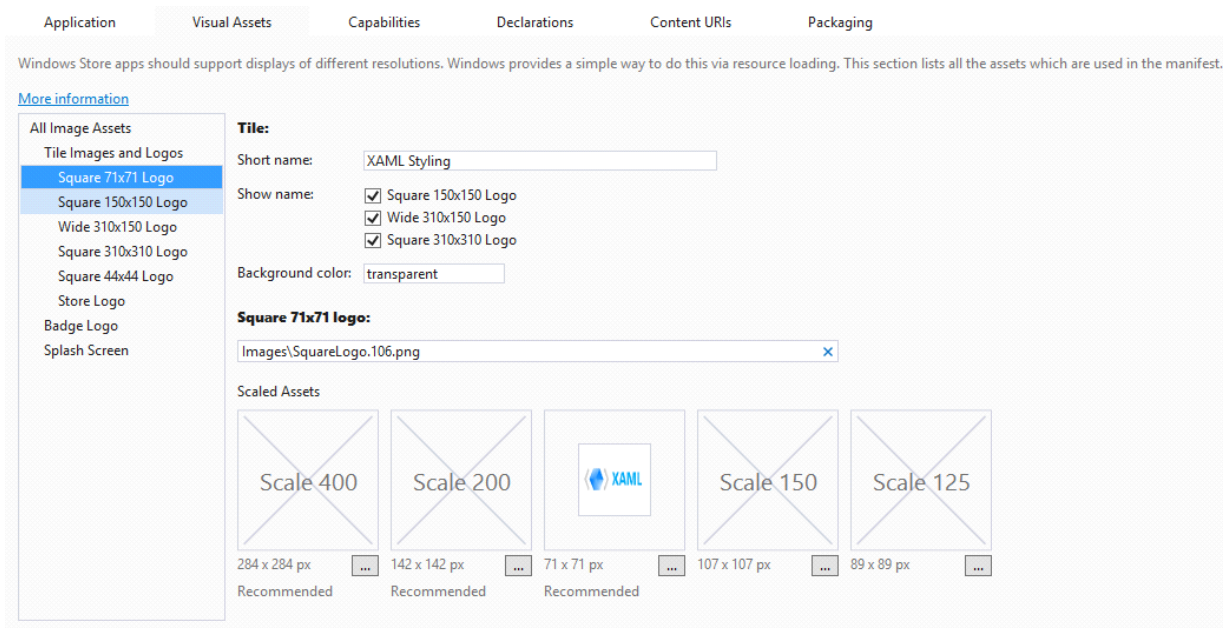


Figure 8

Format of Logo

Logo must be in “png” format and if it’s transparent then it’ll be better for visualizing some information. Locate all these logos one by one. After you’ve changed the logos, save it. If you run the application, it’ll look like this.

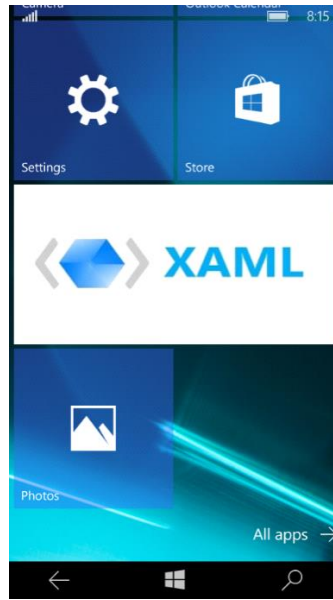
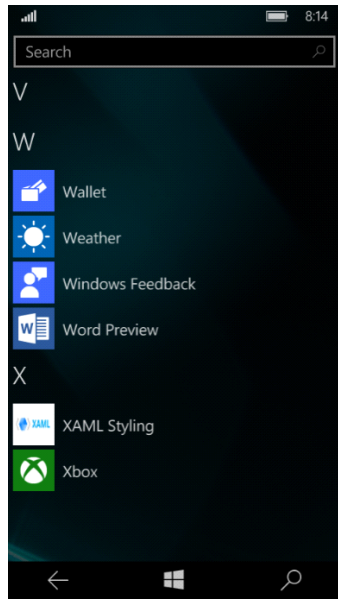


Figure 9.1

Figure 9.2

Figure 9.3

In Windows Store Application, you can find it in All Programs Menu Items. You can also pin it in Start Menu and also can resize it.



Figure 10

Other Options

Well, we've done some major modification to our application. There are some other parts like "Capabilities".

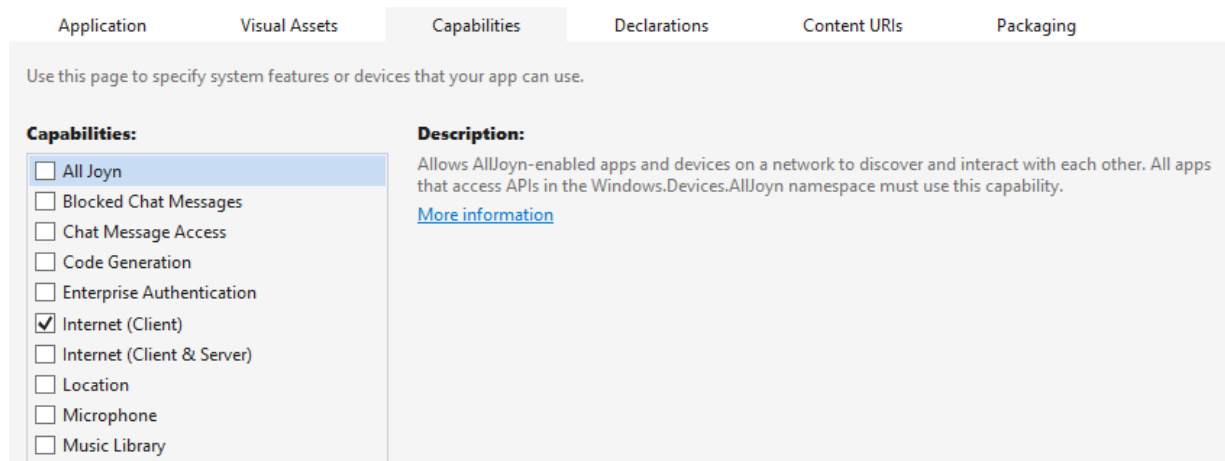


Figure 12

If you use microphone, location service, internet, proximity or any other things, you need to check these options as well. Otherwise your application will not work.

There are some other option like “Declaration”, “Content URLs” and “Packaging”. You can also change these things as well. I have shown the common things here.

So, that’s it. Keep digging with Visual Studio and try to learn everyday new things.

Universal Windows Platform | Extended Splash Screen

Splash Screen is a better way to promote your application and let people have a warm welcome at the start. Universal Windows Platform Application has already built in Splash Screen when you create a new project but this just wiped out in blink of an eye. Hence, people who will use your application can not properly read what is written or the picture was supposed to say. That’s why Extended Splash Screen is very helpful to display what you actually want to say to your user (i.e., what’s this app is about, your credentials, copyright and so on).

Working on a Project

First, create a new blank project or you can use your existing Universal Windows Platform Application in this case. First of all, we are going to add a new folder and give it a name ‘Images’. Now, import your Splash Screen image into this folder. The dimension of the image is recommended to be 620x300 pixel.

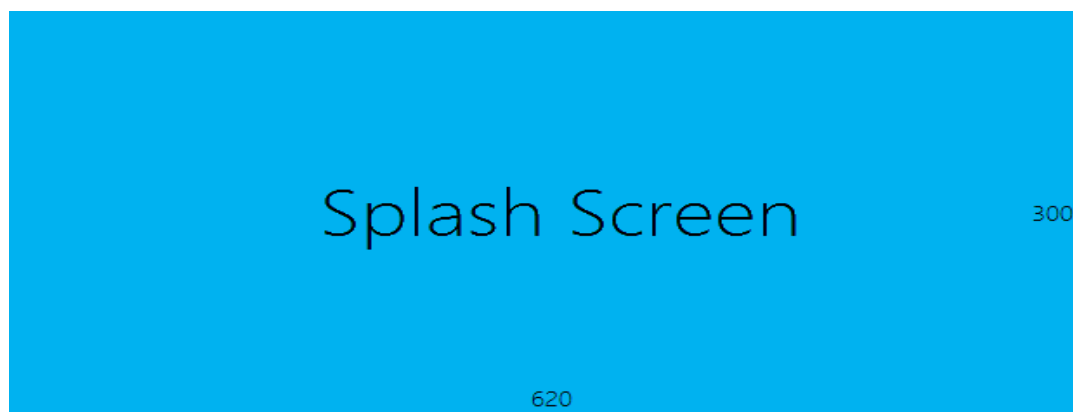


Figure 1: Dimension of Splash Screen.

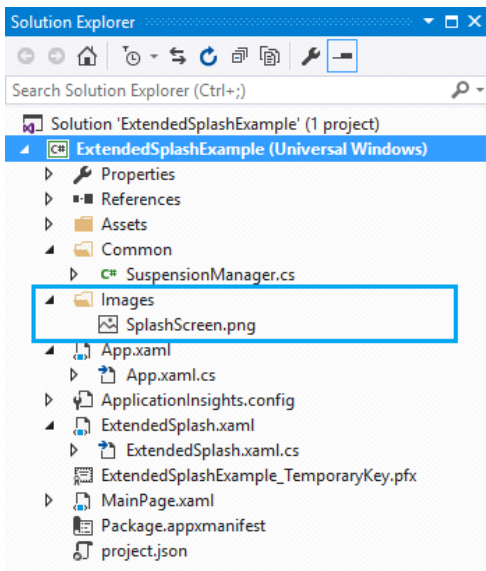


Figure 2: Adding Splash Screen.

After adding the Splash Screen image, go to the 'Package.appxmanifest'. Right click on it and select 'View Code'.

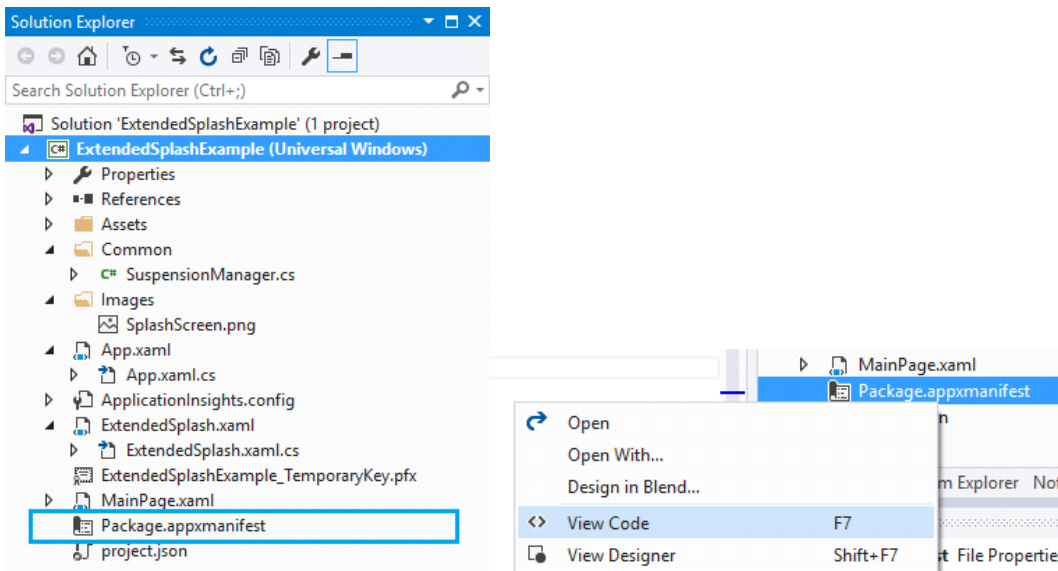


Figure 3: Package.appxmanifest.

After selecting 'View Code', you can see the backend code of the 'appxmanifest' file. Change the Splash Screen image URL with the image that you have added in the 'Images' folder.

```
<Applications>
  <Application Id="App"
    Executable="$targetnametoken$.exe"
    EntryPoint="ExtendedSplashExample.App">
    <uap:VisualElements
      DisplayName="ExtendedSplashExample"
      Square150x150Logo="Assets\Square150x150Logo.png"
      Square44x44Logo="Assets\Square44x44Logo.png"
      Description="ExtendedSplashExample"
      BackgroundColor="transparent">
      <uap:DefaultTile Wide310x150Logo="Assets\Wide310x150Logo.png"/>
      <uap:SplashScreen Image="Images\SplashScreen.png" BackgroundColor="#00B2F0" />
    </uap:VisualElements>
  </Application>
</Applications>
```

Figure 4: Changing Image URL.

In the above picture, you can see that we have set the BackgroundColor in a specific color code. We have matched the color with our Splash Screen's background color and set in the code. You can find the color code by simply opening the image in Visual Studio.

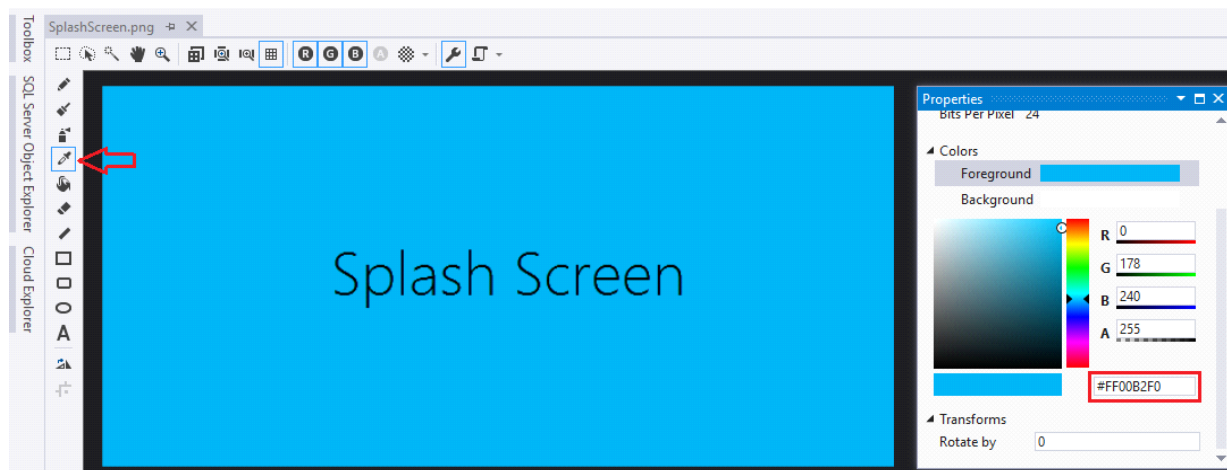


Figure 5: Getting color code.

Select the Dropper tool and click on the Splash Screen background and you can see the actual color code in the Properties menu on the Colors section.

Extended Splash Screen

Now, add a new Blank Page and give a name 'ExtendedSplash'. Modify the main Grid as it is shown in the code below.

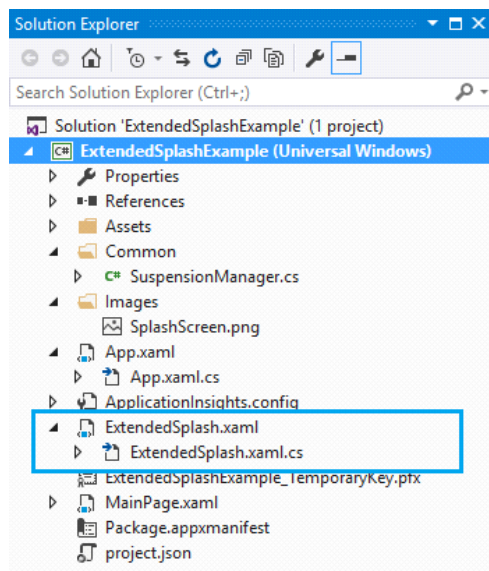


Figure 6: Extended Splash

```
<Page
...
mc:Ignorable="d" Background="#00B2F0">
<Grid Background="#00B2F0">
    <Canvas>
        <Image x:Name="extendedSplashImage" Source="Images/SplashScreen.png" />
    </Canvas>
    <ProgressRing Name="splashProgressRing" IsActive="True" Height="30" Width="30"
        HorizontalAlignment="Center" VerticalAlignment="Bottom" Margin="20" Foreground="White"/>
</Grid>
</Page>
```

Listing: 1

One more thing, you have to do is to add a new helper class, which you can easily find in [Official Windows Platform Sample's Profile](#). Create a new folder named 'Common' and put the .cs file under the folder.

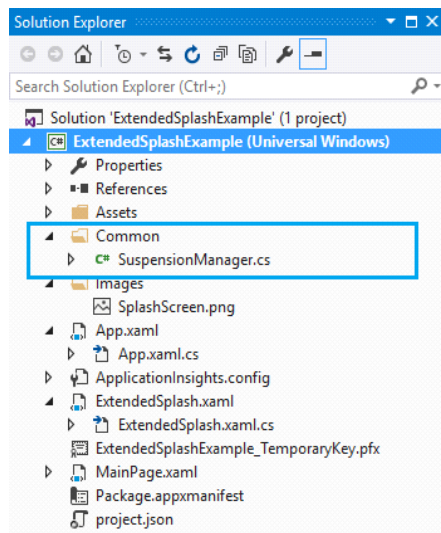


Figure 7: Adding a helper class.

Now come back to the ExtendedSplash.xaml.cs. We have main task to do here. The full source of backend file is given below with description.

```
publicsealedpartialclassExtendedSplash : Page
{
    internalRect splashImageRect; // Rect to store splash screen image coordinates.
    internalbool dismissed = false; // Variable to track splash screen dismissal status.
    internalFrame rootFrame;

    privateSplashScreen splash; // Variable to hold the splash screen object.
    privatedouble ScaleFactor; //Variable to hold the device scale factor (use to determine phone screen resolution)

    public ExtendedSplash(SplashScreen splashscreen, bool loadState)
    {
        InitializeComponent();
        DismissExtendedSplash();

        // Listen for window resize events to reposition the extended splash screen image accordingly.
        // This is important to ensure that the extended splash screen is formatted properly in response to snapping, unsnapping,
        // rotation, etc...
        Window.Current.SizeChanged += newWindowSizeChangedEventHandler(ExtendedSplash_OnResize);

        ScaleFactor = (double)DisplayInformation.GetForCurrentView().ResolutionScale / 100;

        splash = splashscreen;

        if (splash != null)
        {
            // Register an event handler to be executed when the splash screen has been dismissed.
            splash.Dismissed += newTypedEventHandler<SplashScreen, Object>(DismissedEventHandler);

            // Retrieve the window coordinates of the splash screen image.
            splashImageRect = splash.ImageLocation;
            PositionImage();
        }

        // Restore the saved session state if necessary
        RestoreStateAsync(loadState);
    }

    asyncvoid RestoreStateAsync(bool loadState)
    {
        if (loadState)
```

```

        awaitSuspensionManager.RestoreAsync();
    }

    // Position the extended splash screen image in the same location as the system splash screen image.
    void PositionImage()
    {
        extendedSplashImage.SetValue(Canvas.LeftProperty, splashImageRect.Left);
        extendedSplashImage.SetValue(Canvas.TopProperty, splashImageRect.Top);
        extendedSplashImage.Height = splashImageRect.Height / ScaleFactor;
        extendedSplashImage.Width = splashImageRect.Width / ScaleFactor;
    }

    void ExtendedSplash_OnResize(Object sender, WindowSizeChangedEventArgs e)
    {
        // Safely update the extended splash screen image coordinates. This function will be fired in response to
        // snapping, unsnapping, rotation, etc...
        if (splash != null)
        {
            // Update the coordinates of the splash screen image.
            splashImageRect = splash.ImageLocation;
            PositionImage();
        }
    }

    // Include code to be executed when the system has transitioned from the splash screen to the extended splash screen
    // (application's first view).
    void DismissedEventHandler(SplashScreen sender, object e)
    {
        dismissed = true;
    }

    asyncvoid DismissExtendedSplash()
    {
        awaitTask.Delay(TimeSpan.FromSeconds(3)); // set your desired delay
        rootFrame = newFrame();
        MainPage mainPage = newMainPage();
        rootFrame.Content = mainPage;
        Window.Current.Content = rootFrame;
        rootFrame.Navigate(typeof(MainPage)); // call MainPage
    }
}

```

Listing: 2

Here, we have changed the Constructor, which takes two parameters- one is a Splash Screen type and another is a Boolean variable which indicates the loading state of the Splash Screen. There are four major functions, ExtendedSplash_OnResize, PositionImage, RestoreStateAsync and DismissExtendedSplash. ExtendedSplash_OnResize calls the PositionImage method based on the windows size and sets the coordinates. PositionImage sets the Canvas property which depends on the resolution of the screen and DismissExtendedSplash sets the time to display the Splash Screen and navigate to the MainPage.

One last thing, we have to do is working with app.xaml.cs file. Open up the file and change the OnLaunched method accordingly.

```

protectedoverridevoid OnLaunched(LaunchActivatedEventArgs e)
{
    ...
    // Display an extended splash screen if app was not previously running.
    if (e.PreviousExecutionState != ApplicationExecutionState.Running)
    {
        bool loadState = (e.PreviousExecutionState == ApplicationExecutionState.Terminated);
        ExtendedSplash extendedSplash = newExtendedSplash(e.SplashScreen, loadState);
        rootFrame.Content = extendedSplash;
        Window.Current.Content = rootFrame;
    }
    //Window.Current.Content = rootFrame;
    ...
}

```

```
}
```

Listing: 3

We have commented out the current content because it prevents to load the main page and load the Splash Screen page instead. Also, change the OnSuspending method like the one shown below.

```
asyncprivatevoid OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    awaitSuspensionManager.SaveAsync();
    deferral.Complete();
}
```

Listing: 4

We have set all things which are required. Now you are good to go. Build the application, and hopefully it builds successfully. Now run the application and it will work like a charm.

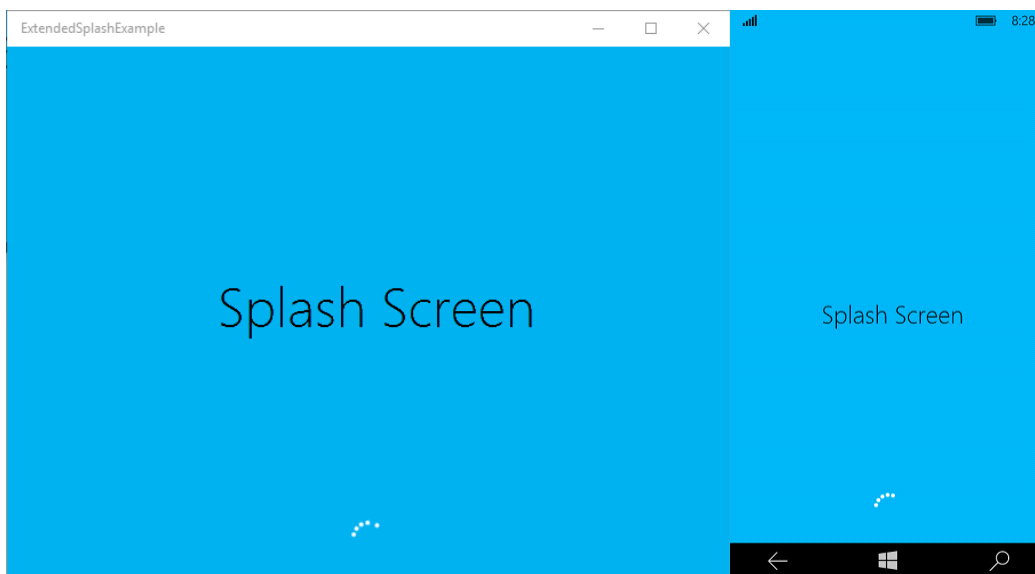


Figure 8: Splash Screen

Closure

Hopefully this will help you to understand the mechanism of Extended Splash Screen. Now, you can make you app more attractive with great promoting Splash Screen imbedded in your application. Happy coding!

Download the project:

UWP | Page navigation and pass a complex object to a page

Previously in Universal Applications, page navigation technique was quite different. In Universal Windows Platform app, it can have several pages and when we want to navigate through different pages, it opens a new window but in Universal Windows Platform, there is only one window. When we want to open a different page, it opens a new frame. So things got changed a little like Windows 10.

So, let's see how it works in Universal Windows Platform page navigation. Let's get started.

Creating a New Project

First, open up a new Blank Project. We need another page to navigate from "MainPage" to the second page. So, we need to add a "Blank Page" template and give it a name "SecondPage.xaml".

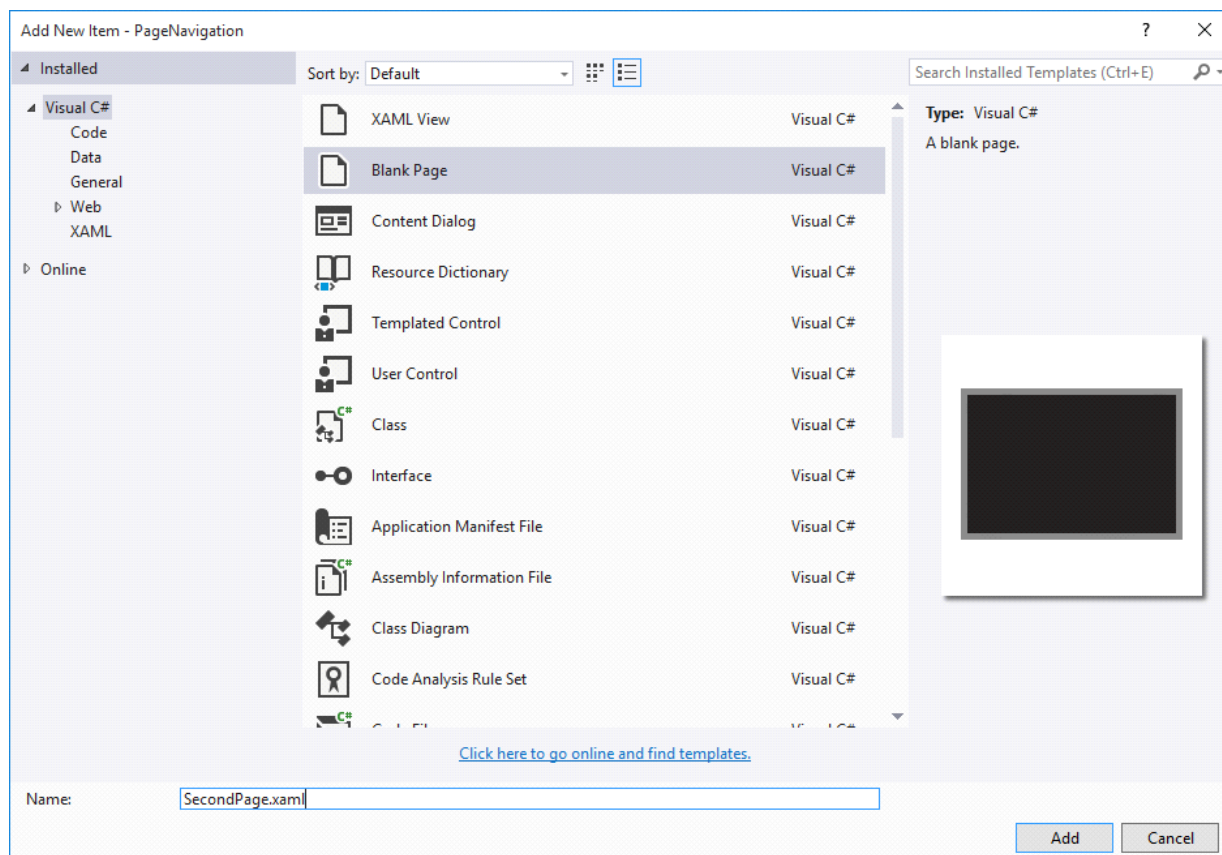


Figure 1

Adding a New Class

Now, we've to add another class name "Person.cs". We'll create two person's object "Name" and "Blog" which we'll pass when we navigate through pages

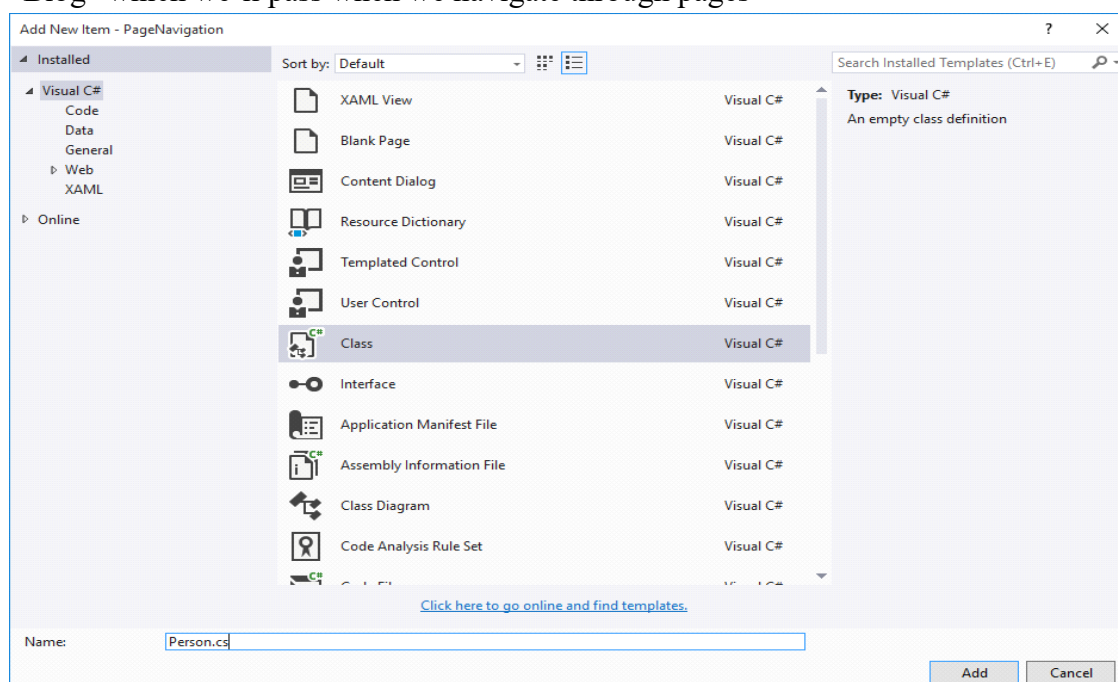


Figure 5

The code is given below.

```

class Person
{
    public string Name { get; set; }
    public string Blog { get; set; }
}

```

Listing 1

Here, we've create to string variable "Name" and "Blog". Just type "prop" and double tab in the keyboard, it will automatically create a full code snippet for you and tab again and change "int" to "string", hit tab second time and change "MyProperty" to "Name". Change the second code snippet likewise.

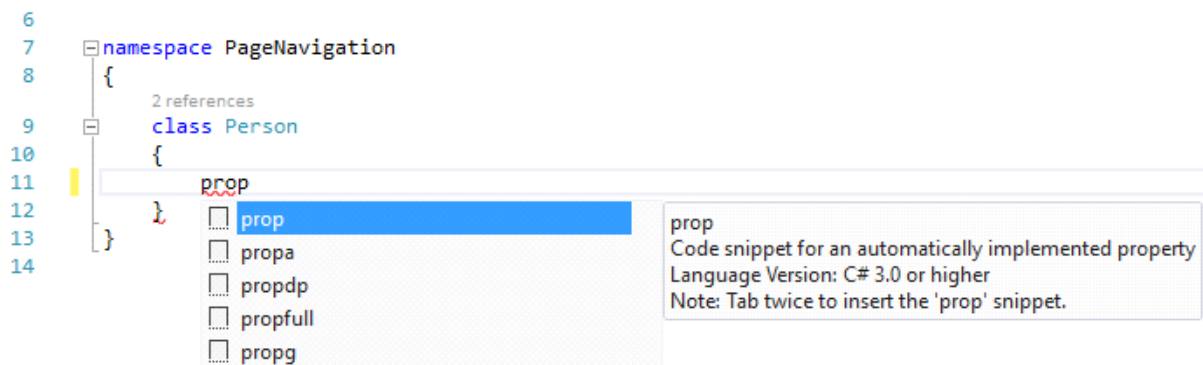


Figure 6

Adding a Button Control

Now, we will work in "MainPage.xaml". Take a Button control, change the content to "Next Page" and the other property like below.

```

<Button Content="Next Page"
        HorizontalAlignment="Left"
        Margin="10,0,0,0"
        VerticalAlignment="Top"
        Click="NextPage_Click"/>

```

Listing 1

The C# code of "MainPage.xaml.cs", mainly the event handler of the Button control is given here.

```

private void NextPage_Click(object sender, RoutedEventArgs e)
{
    var person = new Person { Name = "BD Devs", Blog = "learnwithbddevs.wordpress.com" }
}

```

Listing 2

Here, we have created a person object of our Person class, we've set our "Name" and "Blog" and pass it through the "Second Page". As we mentioned before, Navigation Service does not work in Universal Windows Platform app, it opens up a new frame every time. So, here we call Frame class which has another Navigate Method, which has some types. Like here we have "Basic Page" template and we have passed our person object as a parameter.

Displaying Data in Second Page

Now, in our “SecondPage.xaml” we have taken two TextBlock “tb1” and “tb2”. The XAML code is given below,

```
<TextBlockx:Name="tb1"
    HorizontalAlignment="Left"
    Margin="10,10,0,0"
    TextWrapping="Wrap"
    VerticalAlignment="Top"
    Height="50"
    Width="302"
    FontSize="20"/>
<TextBlockx:Name="tb2"
    HorizontalAlignment="Left"
    Margin="10,65,0,0"
    TextWrapping="Wrap"
    VerticalAlignment="Top"
    Height="50"
    Width="302"
    FontSize="18"/>
```

Listing 3

C# code is given here.

```
protectedoverridevoidOnNavigatedTo(NavigationEventArgs e)
{
    var person = (Person)e.Parameter;

    if(!string.IsNullOrEmpty(person.Name))
    {
        tb1.Text = "Hello, " + person.Name;
        tb2.Text = "Welcome to: " + person.Blog;
    }
    else
    {
        tb1.Text = "Name is required. Go back and enter a name.";
    }
}
```

Listing 4

We have to put all the code in “OnNavigatedTo” method because when a new page loads, this code block will work to retrieve data while navigating. This method is built in and you just need to modify yourself as you like to do.

In our case, we’ve created a person variable object, which we have been parsed into “Person” class. We have checked whether Person’s name is empty or not. If it’s not empty, data will be displayed otherwise not. It will show an error message “Name is required. Go back and enter a name.”.

Adding Back Button

In Universal Windows Platform app, there is no back button in desktop version. Though, phone app has built in back button, if you press the back button it will bring you directly to the Home of your phone. It is not a good User Experience. To avoid this kind of scenario, you can just modify your app.xaml.cs file a little bit. Open your app.xaml.cs file and put the little code inside the OnLaunched method, below rootFrame.NavigationFailed... event.

```
rootFrame.NavigationFailed += OnNavigationFailed;
rootFrame.Navigated += OnNavigated;

// OnNavigated Method
privatevoidOnNavigated(object sender, NavigationEventArgs e)
{
    // Each time a navigation event occurs, update the Back button's visibility
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
        ((Frame)sender).CanGoBack ?
```



```

        AppViewBackButtonVisibility.Visible :
        AppViewBackButtonVisibility.Collapsed;
    }

```

Listing: 5

Moreover, you need to implement a Back Key Request event. To do that, put the code in OnLaunched method, below the Window.Current.Content... event.

```

Window.Current.Content = rootFrame;

SystemNavigationManager.GetForCurrentView().BackRequested += OnBackRequested;

SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
rootFrame.CanGoBack ?
AppViewBackButtonVisibility.Visible :
AppViewBackButtonVisibility.Collapsed;

// OnBackRequested Method
private void OnBackRequested(object sender, BackRequestedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;

    if (rootFrame.CanGoBack)
    {
        e.Handled = true;
        rootFrame.GoBack();
    }
}

```

Listing: 6

Note: Place the OnNavigated&OnBackRequested method outside the OnLaunched method.

All of our work is done. If you run the application, it'll look like this.

Desktop Application:

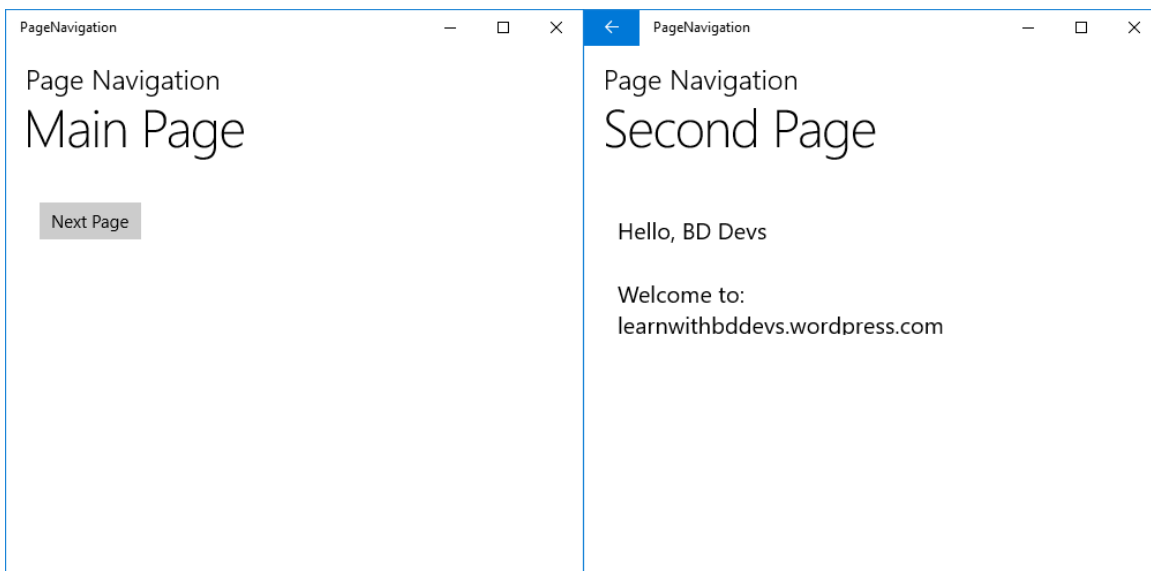


Figure: 7

Mobile Application:

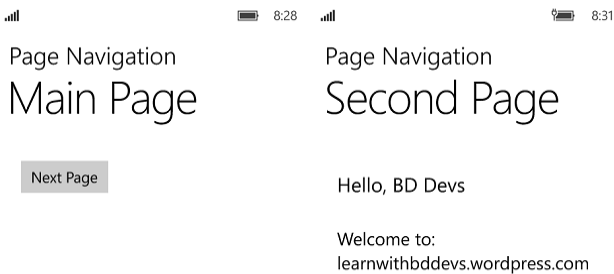


Figure: 8

Hit the “Next Page” Button and it will navigate to the “Second Page” and display the “Name” and “Blog” info on the TextBlocks respectively.

Universal Windows Platform | Command Bar

Command Bar is another user interface element that we can use on the Universal Windows Platform to provide more options in the form of icons and menu options to the user than what is displayed by default .It is really helpful to implement simple navigation and some feature to your app. So, let’s crack in Universal Windows Platform Command Bar.

Creating a New Project and Add a CommandBar

First of all, open up a new project or you can simply load your previous project. Now, in your left-side of Visual Studio, you can see a side window named “Document Outline”. Just right click on the “BottomAppBar” and add a “Command Bar”.

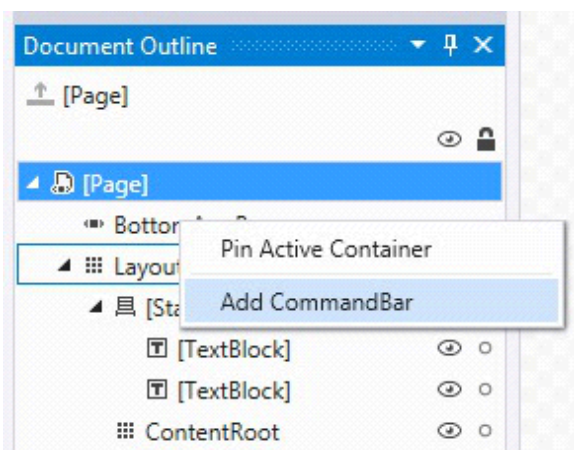


Figure 1

Now, you will notice that, it automatically generates a XAML code snippet for you.

```
<Page.BottomAppBar>
  <CommandBar>
    <AppBarButton Icon="Accept" Label="appbarbutton"/>
    <AppBarButton Icon="Cancel" Label="appbarbutton"/>
  </CommandBar>
</Page.BottomAppBar>
```

</Page.BottomAppBar>

Listing 1

Now, again right click on the “SecondaryCommands” and add a new “AppBarButton”.

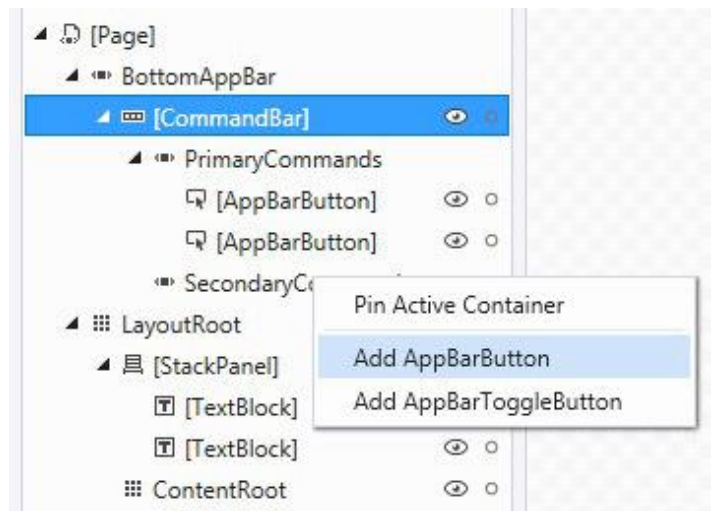


Figure 2

You can see that it creates another menu button for your “Command Bar” and your XAML code will look like this.

```
<Page.BottomAppBar>
  <CommandBar>
    <CommandBar.SecondaryCommands>
      <AppBarButton Label="about"/>
    </CommandBar.SecondaryCommands>
    <AppBarButton Label="accept" Icon="Accept"/>
    <AppBarButton Label="cancel" Icon="Cancel"/>
  </CommandBar>
</Page.BottomAppBar>
```

Listing 2

We’ve modified the labels of “Accept” and “Cancel” button, it shows the hint about the “Command Bar” buttons, and changed the label of “AppBarButton” to “about”.

Changing the CommandBar Icons

Now, you can change the icons of your “Command Bar” as you see in the picture below, it is already set to “Accept” icon because we’ve selected the “accept” button in the XAML code.

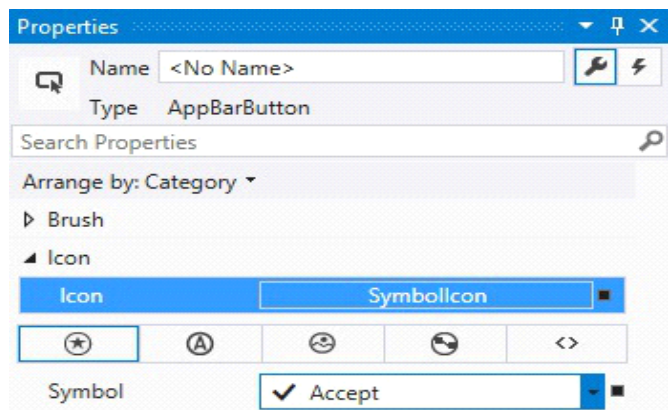


Figure 3

Or you can choose a “font icon” as your Icon, like “A”, “B” or anything you want.

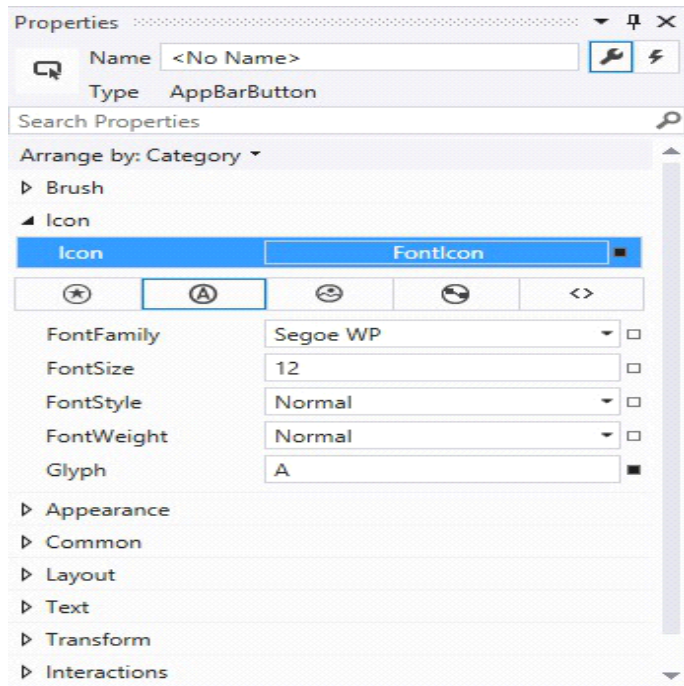


Figure 4

If you change “Accept” to “A” and “Cancel” to “C”, the code will look like this.

```
<Page.BottomAppBar>
  <CommandBar>
    <CommandBar.SecondaryCommands>
      <AppBarButton Label="about"/>
    </CommandBar.SecondaryCommands>
    <AppBarButton Label="accept">
      <AppBarButton.Icon>
        <FontIcon Glyph="A"/>
      </AppBarButton.Icon>
    </AppBarButton>
    <AppBarButton Label="cancel">
      <AppBarButton.Icon>
        <FontIcon Glyph="C"/>
      </AppBarButton.Icon>
    </AppBarButton>
  </CommandBar>
</Page.BottomAppBar>
```

Listing 3

Changing the CommandBar Mode

You can also change the mode of “Command Bar”. For this, all you’ve to do, is just make some change in your “” tag like the picture below.



Figure 5

You’ve to select the “Minimal” view of “ClosedDisplayMode”.

```

<Page.BottomAppBar>
  <CommandBarClosedDisplayMode="Minimal">
    <CommandBar.SecondaryCommands>
      <AppBarButton Label="about"/>
    </CommandBar.SecondaryCommands>
    <AppBarButton Label="accept">
      <AppBarButton.Icon>
        <FontIcon Glyph="A"/>
      </AppBarButton.Icon>
    </AppBarButton>
    <AppBarButton Label="cancel">
      <AppBarButton.Icon>
        <FontIcon Glyph="C"/>
      </AppBarButton.Icon>
    </AppBarButton>
  </CommandBar>
</Page.BottomAppBar>

```

Listing 4

Add an Event Handler

Now let’s go back to the previous changes and give an event handler in “about” button. In XAML code, select the “About AppBarButton” line and under the “Solution Explorer”, you can see a “Properties” window and select the thunder sign like the picture below:

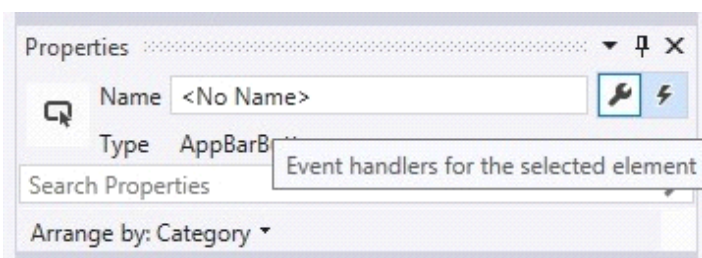


Figure 6

Double click on the “Click” section and it automatically generates the code block for you.

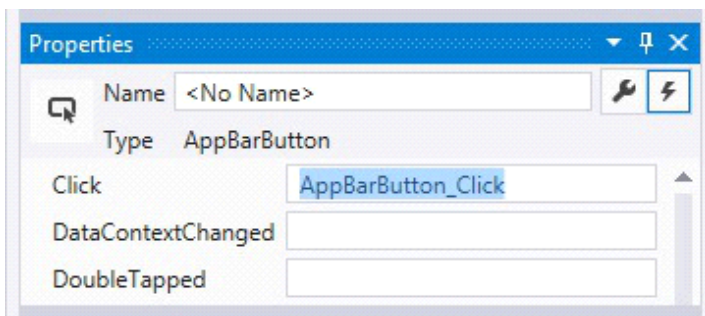


Figure 7

If you take a look at the XAML code, you can see exactly like this.

```
<Page.BottomAppBar>
  <CommandBarClosedDisplayMode="Minimal">
    <CommandBar.SecondaryCommands>
      <AppBarButton Label="about" Click="AppBarButton_Click"/>
    </CommandBar.SecondaryCommands>
    <AppBarButton Label="accept" Icon="Accept"/>
    <AppBarButton Label="cancel" Icon="Cancel"/>
  </CommandBar>
</Page.BottomAppBar>
```

Listing 5

As, we have made an event handler for our “about” button, so we have to take another page named “AboutPage.xaml” and we will navigate to that page if someone taps on the “about” button.

So, when you have done adding a new page, go to the “MainPage.xaml.cs” or wherever you have done this. In our case, we created our “Command Bar” in “MainPage.xaml”, and you can see this code block in that page.

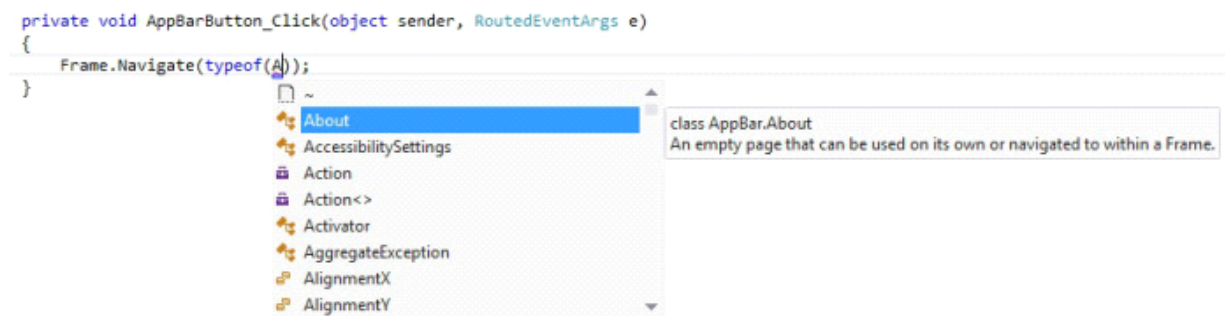


Figure 8

When we are done, the code will look like this.

```
private void AppBarButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(AboutPage));
}
```

Listing 6

Running the Application

After all you set, run the application and it will look like this.

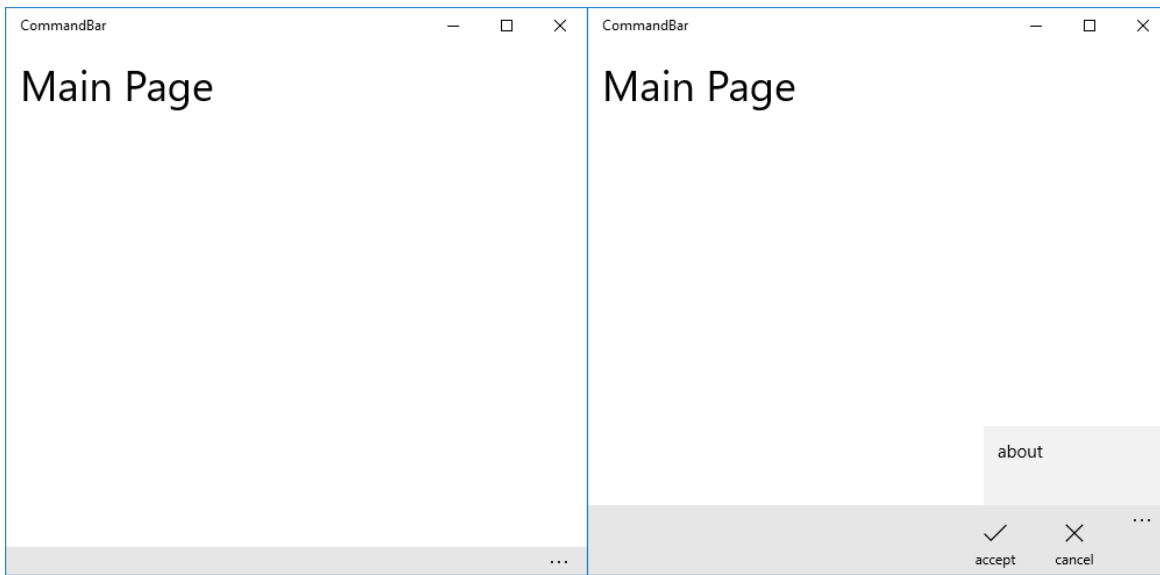


Figure: 9

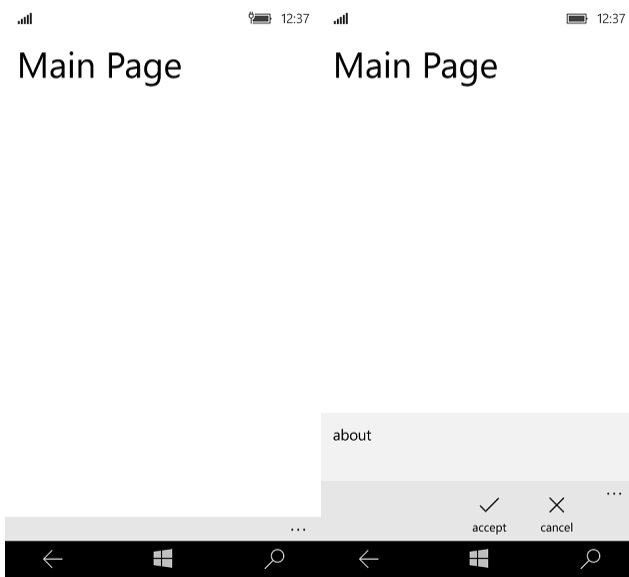


Figure: 10

When you tap on the “about” button, it navigates to the “AboutPage.xaml” page.

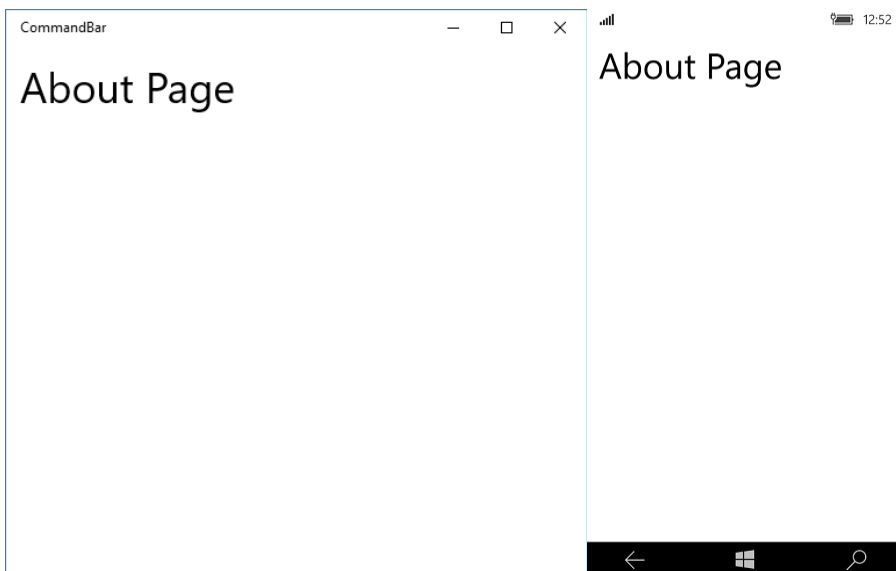


Figure: 11

Closure

So, that's it. I think you got the basic idea of how to use "Command Bar" in Universal Windows Platform Application. Have a nice day. Happy coding!

Universal Windows Platform | Data Binding

Data binding is a useful technique to populate data into your application. It's really useful when you have massively structured code and you've to handle a lot of data. It binds your whole bunch of data into a single control like ListView, ListBox, FlipView or it can be your custom control. XAML gives us the power to bind the data into these structural controls with less effort and in an efficient way.

So let's get started with Universal Windows Platform app development with a simple data binding technique.

Different Binding Modes

There are a number of different binding modes, as defined in the Binding Mode enumeration:

TwoWay – Changes to the UI or model automatically updates the other. This is used for editable forms or other interactive scenarios.

OneWay – Updates the UI when the model changes. This is appropriate for read-only controls populated from data.

OneTime – Updates the UI when the application starts or when the data context changes. This is used when the source data is static/does not change.

OneWayToSource – Changes to the UI update the model

Default – Uses the default mode for the particular binding target (UI control). This differs based on the control.

Universal Windows Platform introduces a new kind of binding which is **Compile Binding (x:Bind)**. Comparing traditional binding techniques, it is really efficient and also faster. If you have less amount of data, maybe you won't notice the actual change of execution time but when you're working on vast amount of data, you will notice the actual differences. To start with our project, first create a new blank application. Subsequently, create a new folder and give it a name "Images". I have used [Images](#) from Microsoft theme download site. Now, add these images into this folder.

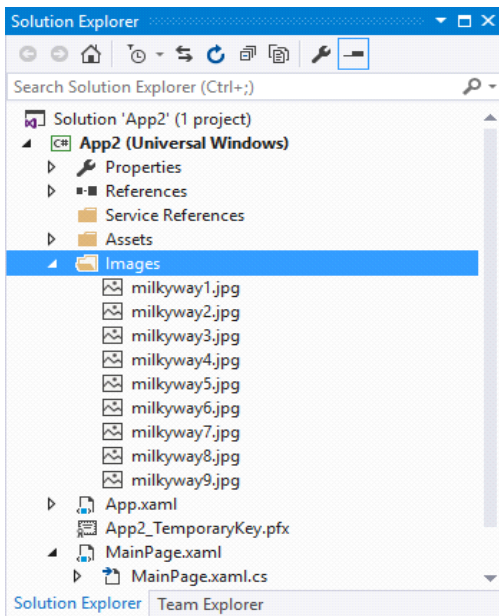


Figure: 1

Designing UI

To design the UI element, we have used FlipView control. Since we want to show the images by using databinding technique, paste the code into your main grid element.

```
<FlipView Name="MyFlipView" BorderBrush="Black" BorderThickness="1" ItemsSource="{x:Bind Items}"
xmlns:m="using:App2">
  <FlipView.ItemTemplate>
    <DataTemplate x:DataType="m:Images">
      <Grid>
        <Image Source="{x:Bind images}" Stretch="Fill" />
        <Border Background="#A5000000" Height="80" VerticalAlignment="Bottom" >
          <TextBlock Text="{x:Bind title}" FontFamily="Segoe UI" FontSize="26.667"
Foreground="#CCFFFFFF" Padding="15,20" />
        </Border>
      </Grid>
    </DataTemplate>
  </FlipView.ItemTemplate>
</FlipView>
```

Listing: 1

Here, the name of the FlipView control is “MyFlipView” and there are two main child elements: Image and TextBox. TextBox control is wrapped by a Border control which gives a little bit of effect in the UI.

Most importantly, if you notice we’ve bind the ItemSource by compile binding and that’s why we’ve to define DataType (m:Images) by its class name. Here Images is the name of the class which contains an image and sting attribute.

```
publicclass Images
{
    publicBitmapImage images { get; set; }
    publicstring title { get; set; }
}
```

Listing: 2

Lastly, we bind the Image and TextBox with “images” and “title”. The backend code will be like this.

```
publicObservableCollection<Images> Items { get; private set; } = newObservableCollection<Images>();
```

...

```

BitmapImage[] bitmapImage = newBitmapImage[9];
bitmapImage[0] = newBitmapImage(newUri("ms-appx:///Images/milkyway1.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[1] = newBitmapImage(newUri("ms-appx:///Images/milkyway2.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[2] = newBitmapImage(newUri("ms-appx:///Images/milkyway3.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[3] = newBitmapImage(newUri("ms-appx:///Images/milkyway4.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[4] = newBitmapImage(newUri("ms-appx:///Images/milkyway5.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[5] = newBitmapImage(newUri("ms-appx:///Images/milkyway6.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[6] = newBitmapImage(newUri("ms-appx:///Images/milkyway7.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[7] = newBitmapImage(newUri("ms-appx:///Images/milkyway8.jpg", UriKind.RelativeOrAbsolute));
bitmapImage[8] = newBitmapImage(newUri("ms-appx:///Images/milkyway9.jpg", UriKind.RelativeOrAbsolute));

string[] title = newstring[] { "Milkyway 1", "Milkyway 2", "Milkyway 3", "Milkyway 4", "Milkyway 5", "Milkyway 6",
"Milkyway 7", "Milkyway 8", "Milkyway 9" };

for (int i = 0; i < 9; i++)
{
    this.Items.Add(newImages { images = bitmapImage[i], title = title[i] });
}

MyFlipView.ItemsSource = Items;

```

Listing: 3

Here, we declare a list item named “Items” and assign it to “MyFlipView” control. We have added the image and title array to the collection list by using a for loop.

If we run the application, it will look like this.

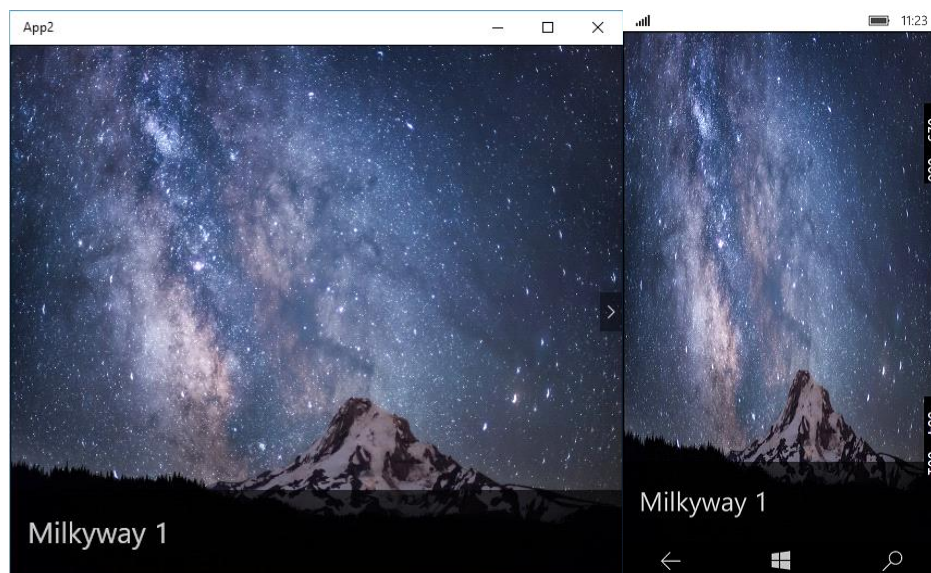


Figure: 2

Another way to bind is a traditional way is what we were doing before in Windows Phone 8.1 and later. We have changed our XAML code a little bit.

```

<FlipView Name="MyFlipView" BorderBrush="Black" BorderThickness="1" ItemsSource="{Binding Items}">
    <FlipView.ItemTemplate>
        <DataTemplate>
            <Grid>
                <Image Source="{Binding images}" Stretch="UniformToFill" />
                <Border Background="#A5000000" Height="80" VerticalAlignment="Bottom" >
                    <TextBlock Text="{Binding title}" FontFamily="Segoe UI" FontSize="26.667"
                        Foreground="#CCFFFFFF" Padding="15,20" />
                </Border>
            </Grid>
        </DataTemplate>
    </FlipView.ItemTemplate>
</FlipView>

```

Listing: 4

If we compare two binding techniques, we can see that compile binding is much faster than normal binding.

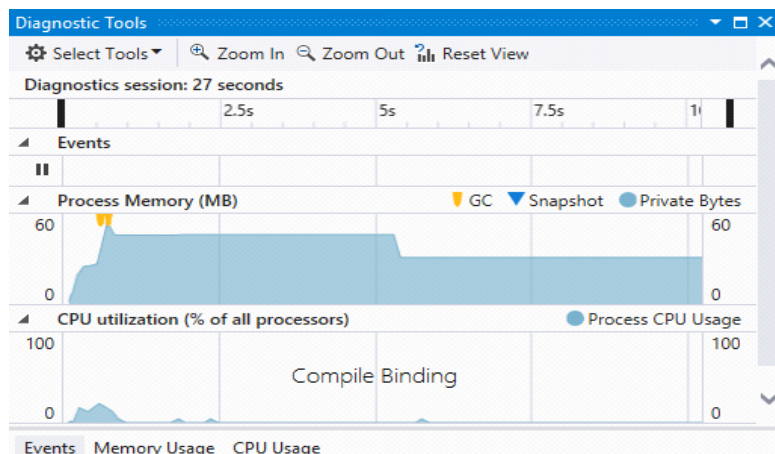


Figure 3: Compile Binding

Here the memory process is 60 bytes. If you look at the normal binding, the memory process is 70 bytes.

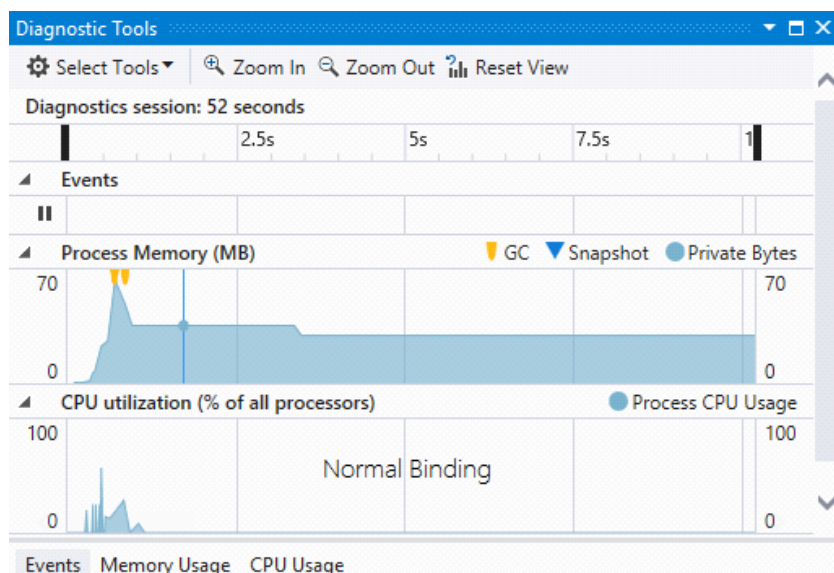


Figure 4: Normal Binding

Closure

Hopefully, you have understood the binding techniques in Universal Windows Platform app development. You can apply this procedure in any form of application and make an efficient way to process your data. Happy coding.

Universal Windows Platform | MVVM

“MVVM” stands for “Model-View-ViewModel”. Model basically initialize properties, attributes whatever you say and “ViewModel” contains all data of your application. Finally View represents actual representation of data in your screen. Basically Data Binding works between “ViewModel”, “View” layer, “View” requests command, “ViewModel” accepts it and responses to the “View”. I’m not going the whole theoretical knowledge. I tried to give you the basic idea what “MVVM” is.

Now, let’s get crack in the best practice in this super awesome architectural model.

Creating a New Project

First of all, open up a new project and name it “MVVMEEx” or whatever you want. Now, add three folders “Models”, “ViewModels” and “Views” one by one, like this.

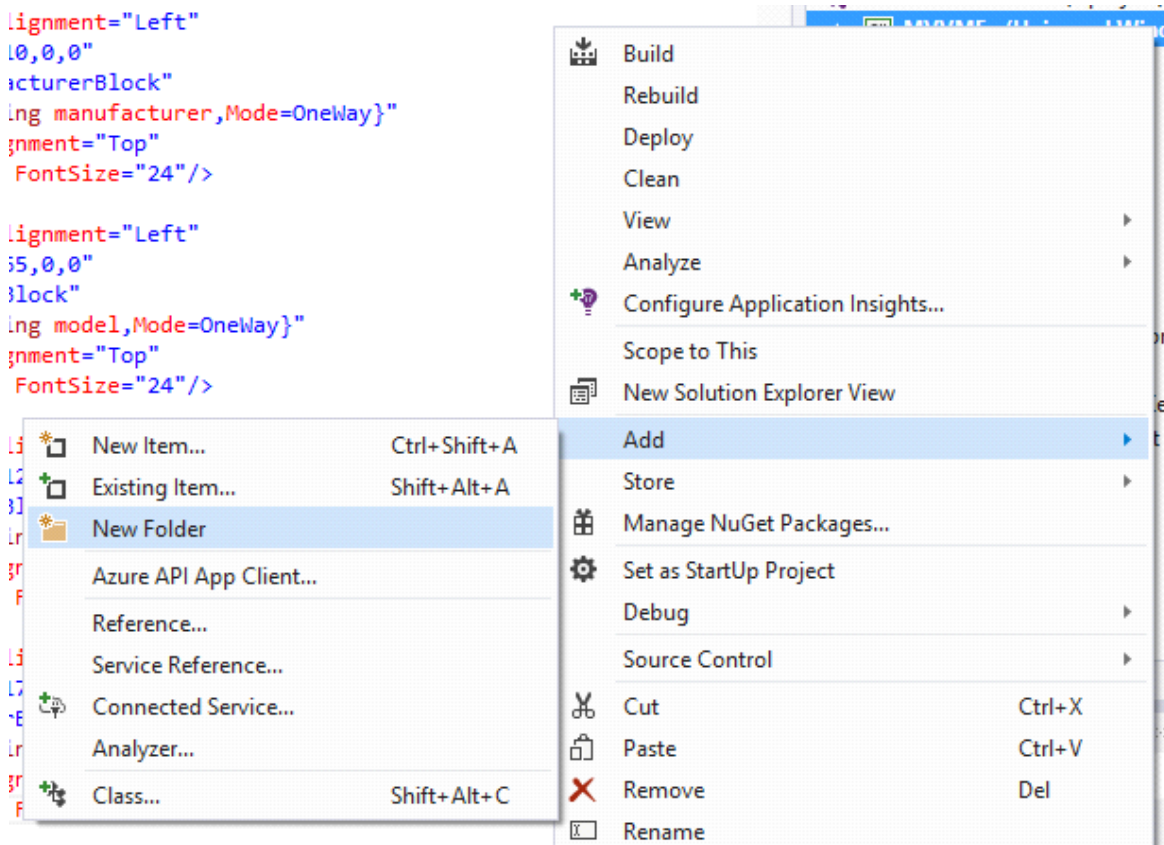


Figure 1

It should look exactly like below in Figure 2.

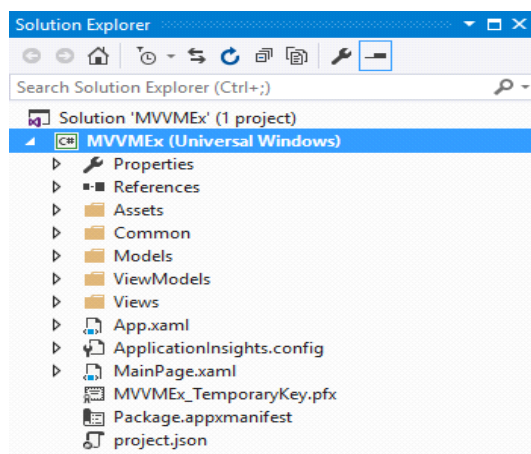


Figure 2

Adding a New Page

Now, in “Views” folder, right click and add a new “Blank Page”, give it a name “AutoView”.

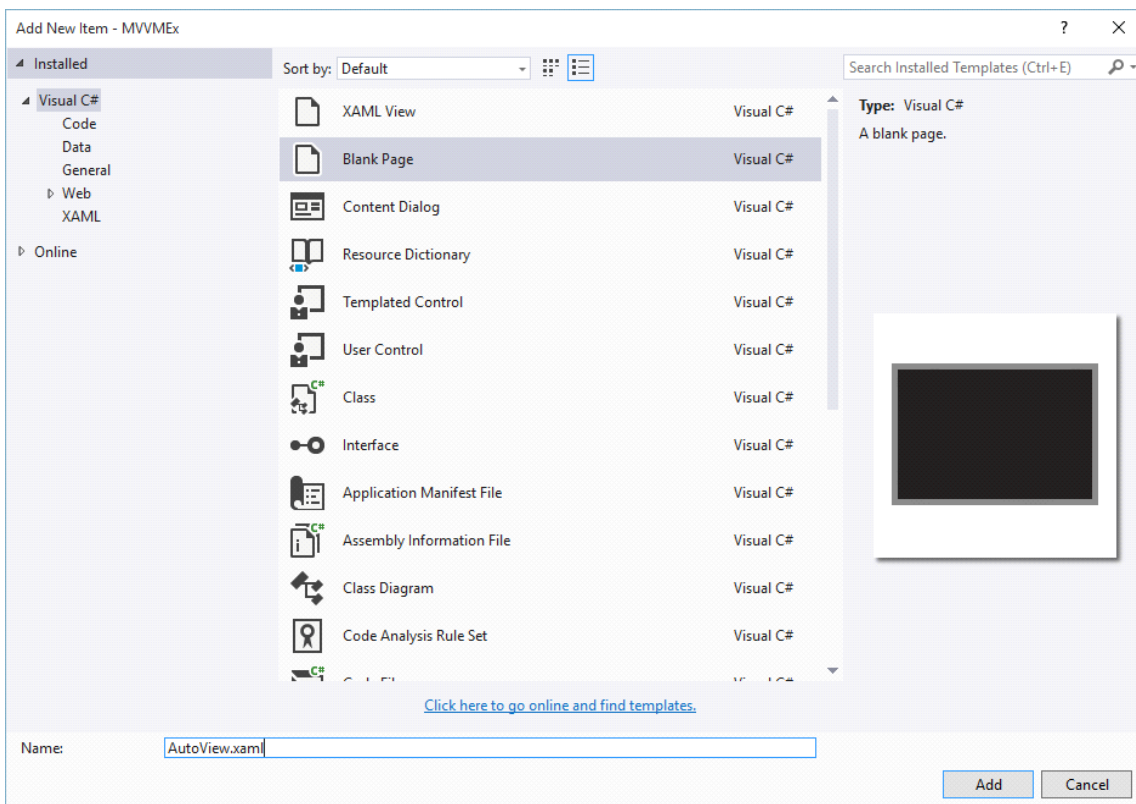


Figure 3

Changing the Starting Page

Now one more thing we have to do, is changing our starting page. For that, you have to go “app.xaml.cs” and change this line of code

```
if (rootFrame.Content == null)
{
    // When the navigation stack isn't restored navigate to the first page,
    // configuring the new page by passing required information as a navigation
    // parameter
    rootFrame.Navigate(typeof(Views.AutoView), e.Arguments);
}
```

Listing: 1

We have replaced our “MainPage.xaml” and add a new page “AutoView.xaml”.

Adding Classes

Now, similarly right click on the “Model” folder and add a new class named “Auto.cs”. Again right click on the “ViewModels” folder and add another class named “AutoViewModel.cs”. After all your setup, your Solution Explorer will look like in Figure 4.

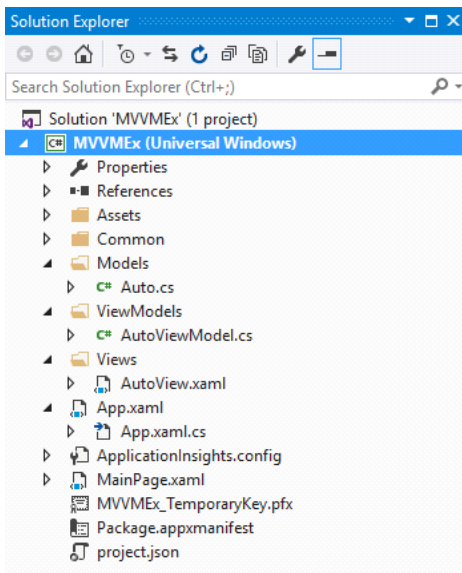


Figure 4

Now we'll modify our "AutoView.xaml" as follows.

Modifying "AutoView.xaml" Page

Setting up app title and information.

```
<!-- Title Panel -->
<StackPanel Grid.Row="0" Margin="19,0,0,0">
  <TextBlock Text="Learn With BD Devs" Style="{ ThemeResource TitleTextBlockStyle}" Margin="0,12,0,0"/>
  <TextBlock Text="MVVM" Margin="0,-6.5,0,26.5" Style="{ ThemeResource HeaderTextBlockStyle}"
    CharacterSpacing="{ ThemeResource PivotHeaderItemCharacterSpacing}"/>
</StackPanel>
```

Listing 2

Modifying main grid.

```
<!--TODO: Content should be placed within the following grid-->
<Grid Grid.Row="1" x:Name="ContentRoot" Margin="19,9.5,19,0">
  <TextBlock Height="50"
    Width="342" FontSize="24"/>
  <TextBlock Height="50"
    HorizontalAlignment="Left"
    Margin="10,65,0,0"
    Name="modelBlock"
    Text="{Binding model,Mode=OneWay}"
    VerticalAlignment="Top"
    Width="342" FontSize="24"/>
  <TextBlock Height="50"
    HorizontalAlignment="Left"
    Margin="10,120,0,0"
    Name="colorBlock"
    Text="{Binding color,Mode=OneWay}"
    VerticalAlignment="Top"
    Width="342" FontSize="24"/>
  <TextBlock Height="50"
    HorizontalAlignment="Left"
    Margin="10,175,0,0"
    x:Name="yearBlock"
    Text="{Binding year, Mode=OneWay}"
    VerticalAlignment="Top"
    Width="342" FontSize="24"/>
</Grid>
```

Listing 3

Implementation of "BaseModel.cs" Class

Now, we'll move to our "Models" folder and initialize auto's properties, but before that, we have to add another class name "BaseModel.cs" in our "Common" folder.

```
publicclassBaseModel : INotifyPropertyChanged
{
    publiceventPropertyChangedEventHandler PropertyChanged;

    protectedbool SetProperty<T>(ref T storage, T value, [CallerMemberName] string propertyName = null)
    {
        if (object.Equals(storage, value)) returnfalse;
        storage = value;
        this.OnPropertyChaned(propertyName);
        returntrue;
    }

    privatevoid OnPropertyChaned(string propertyName)
    {
        var eventHandler = this.PropertyChanged;
        if (eventHandler != null)
            eventHandler(this, newPropertyChangedEventArgs(propertyName));
    }
}
```

Listing 4

It's our "INotifyPropertyChanged" interface. As, we have said in best practice, you have made your code as much clean as you can.

Implementation of "Auto.cs" Class

Now, move back to our "Auto.cs" class. The initialized properties are given below.

```
publicclassAuto : BaseModel
{
    privatestring _manufacturer;
    privatestring _model;
    privatestring _color;
    privateint _year;

    publicstring manufacturer
    {
        get { return _manufacturer; }

        set { this.SetProperty(refthis._manufacturer, value); }
    }

    publicstring model
    {
        get { return _model; }

        set { this.SetProperty(refthis._model, value); }
    }

    publicstring color
    {
        get { return _color; }

        set { this.SetProperty(refthis._color, value); }
    }

    publicint year
    {
        get { return _year; }

        set { this.SetProperty(refthis._year, value); }
    }
}
```

Listing 5

Here, we have inherited all the public properties of “BaseModel.cs” class and fire the value of data in setter. Just a simple logic of OOP (Object Oriented Programming).

Setting Up Data in “AutoViewModel.cs” Class

Now, we set the data of our “Auto” properties in “AutoViewModel.cs” class.

```
public class AutoViewModel : Auto
{
    Auto _auto = new Auto
    {
        manufacturer = "Oldsmobile",
        model = "Cutlas Supreme",
        color = "Silver",
        year = 1988
    };

    public AutoViewModel()
    {
        this.manufacturer = _auto.manufacturer;
        this.model = _auto.model;
        this.color = _auto.color;
        this.year = _auto.year;
    }
}
```

Listing 6

Here, we have used Inheritance and inherited “Auto.cs” class like before so that we can access all the public properties of “Auto.cs” class.

We created a “_auto” object of “Auto.cs” class and initialize all the values here and in constructor “AutoViewModel” we make references of these properties.

Setting Up DataContext in “AutoView.xaml.cs” Class

Our work is almost done. Now, to visualize the data of our “AutoViewModel.cs” class, we have to instantiate in our “AutoView.xaml.cs” class. To do so, change the code as follows.

```
private AutoViewModel defaultViewModel = new AutoViewModel();

public AutoView()
{
    this.InitializeComponent();
    ...
    this.DataContext = defaultViewModel;
}

public AutoViewModel DefaultViewModel
{
    get { return this.defaultViewModel; }
}
```

Listing 7

Here, we have created a “defaultViewModel” object of “AutoViewModel.cs” class and make it the “DataContext” of our “AutoView.xaml.cs” class in constructor. It actually retrieves all the data from “AutoViewModel” constructor and shows in “ContentRoot” grid of “AutoView.xaml”.

Running the Application

Now, it’s time to build our project. After you run the application, it should look exactly like this.

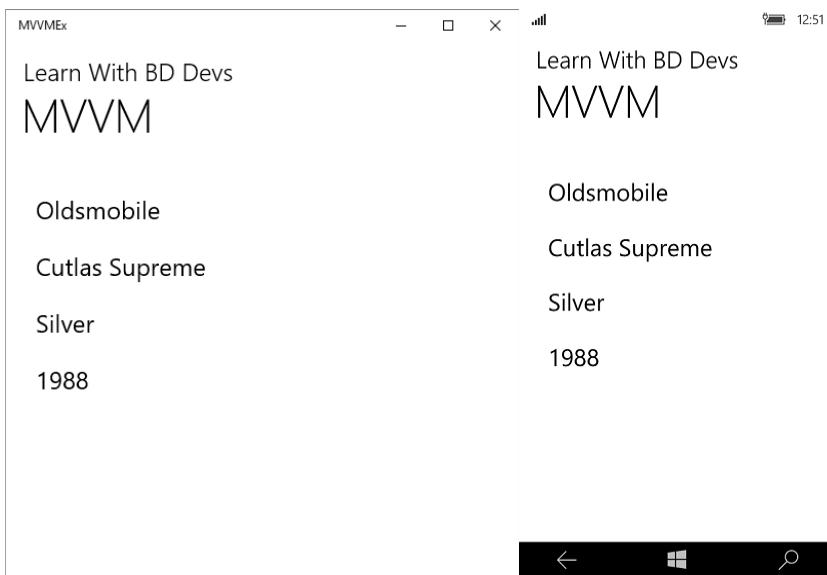


Figure: 5

Closure

Hope you've understood the concept of "MVVM" and how to implement in your project. It's really helpful when you'll work in big project and you've to handle lots of data.

Universal Windows Platform | Map Control

In new update of Universal Windows Platform, there are significant APIs that changed from previous Windows Phone 8.0 & 8.1. An amazing feature like Geofencing is added. So let's crack a whole new Universal Windows PlatformMap Control.

Creating a New Project and Add a Map Control

Take a new project and give it a name simply "Map" or whatever you want. Now, we will add our map control from the "Toolbox". Drag the "MapControl" tool from the "Toolbox" and place it in the main Grid.

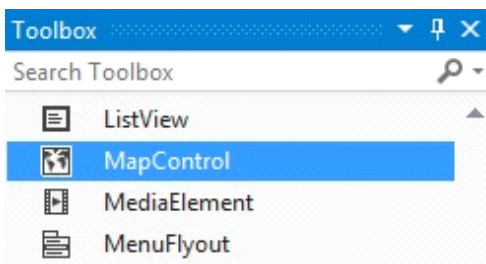


Figure 1

Making Grids

Now, we will modify our main Grid to resize the "MapControl" and to show some other stuff. First of all, we will make some rows to put other controls.

```
<Grid.RowDefinitions>
    <RowDefinition Height="50"/>
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

Listing 1

There are two "RowDefinitions". First one is 50px in height, other is "*" which will cover the rest of the grid space.

Adding Controls

Here are the controls that we have used in our “MainPage.xaml”.

```
<TextBlockTextWrapping="NoWrap"Grid.Row="0"
  Text="Map control"
  FontSize="36" />

<Maps:MapControlGrid.Row="1"
  x:Name="MyMap"
  HorizontalAlignment="Stretch"
  VerticalAlignment="Stretch"
  Width="360"
  Height="590"ZoomLevelChanged="MyMap_ZoomLevelChanged"/>

<BorderCornerRadius="10"x:Name="border"Grid.Row="1"
  Height="70"
  Width="70"
  Canvas.Left="150"
  Canvas.Top="270"
  Background="{ ThemeResourceAppBarItemPointerOverBackgroundThemeBrush }">
  <ProgressRingx:Name="progressRing"
  IsActive="True"
  Background="Transparent"
  Height="40"
  Width="40"/>
</Border>

<SliderGrid.Row="1"x:Name="mySlider" Orientation="Vertical"HorizontalAlignment="Right" Height="211" Width="45"
  Margin="0,0,10,0" Minimum="10" Maximum="20" Value="0"ValueChanged="ZoomValueChanged"
  Foreground="SteelBlue"/>
```

Listing 2

We have used a “ProgressBar” control to indicate while Map is loading, a “TextBlock” will show the title, a “MapControl” to show the Bing Map and a “Slider” to zoom in and zoom out.

Adding AppBar

We have also used a “BottomAppBar” to locate your current position.

```
<Page.BottomAppBar>
  <CommandBarClosedDisplayMode="Minimal" Opacity="0.5">
    <AppBarButton Label="locate me" Icon="Target" Click="LocateMe_Click" />
  </CommandBar>
</Page.BottomAppBar>
```

Listing 3

I’m not going through the details of adding controls and how they work. Please feel free to take a look on my previous articles.

So, finally our design will look like this.

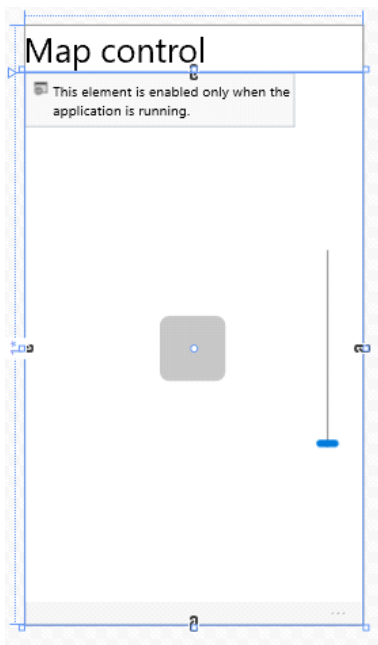


Figure 2

Code Behind

Let's open "MainPage.xaml.cs" and find "OnNavigatedTo" method. Modify it like below.

```
publicsealedpartialclassMainPage :Page
{
    Geolocatorgeolocator;

    publicMainPage()
    {
        this.InitializeComponent();
        this.Loaded += MainPage_Loaded;
    }

    privateasyncvoidMainPage_Loaded(object sender, RoutedEventArgs e)
    {
        // Map Token for testing purpose,
        // otherwise you'll get an alert message in Map Control
        MyMap.MapServiceToken = "abcdef-abcdefghijklmno";

        geolocator = newGeolocator();
        geolocator.DesiredAccuracyInMeters = 50;

        try
        {
            // Getting Current Location
            Geopositiongeoposition = awaitgeolocator.GetGeopositionAsync(
                maximumAge: TimeSpan.FromMinutes(5),
                timeout: TimeSpan.FromSeconds(10));

            MapIconmapIcon = newMapIcon();
            // Locate your MapIcon
            mapIcon.Image = RandomAccessStreamReference.CreateFromUri(newUri("ms-appx:///Assets/my-position.png"));
            // Show above the MapIcon
            mapIcon.Title = "Current Location";
            // Setting up MapIcon location
            mapIcon.Location = newGeopoint(newBasicGeoposition()
            {
                //Latitude = geoposition.Coordinate.Latitude, [Don't use]
                //Longitude = geoposition.Coordinate.Longitude [Don't use]
                Latitude = geoposition.Coordinate.Point.Position.Latitude,
                Longitude = geoposition.Coordinate.Point.Position.Longitude
            });
            // Positon of the MapIcon
            mapIcon.NormalizedAnchorPoint = newPoint(0.5, 0.5);
            MyMap.MapElements.Add(mapIcon);
        }
    }
}
```

```

// Showing in the Map
await MyMap.TrySetViewAsync(mapIcon.Location, 18D, 0, 0, MapAnimationKind.Bow);

// Disable the Progress Bar
progressBar.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
// Set the Zoom Level of the Slider Control
mySlider.Value = MyMap.ZoomLevel;
}
catch (UnauthorizedAccessException)
{
    MessageBox("Location service is turned off!");
}
base.OnNavigatedTo(e);
}
...
}

```

Listing 4

Our main work is done, now we have to write the other methods to work it perfectly.

```

// Slider Control
private void ZoomValueChanged(object sender, RangeBaseValueChangedEventArgs e)
{
    if (MyMap != null)
        MyMap.ZoomLevel = e.NewValue;
}

private void MyMap_ZoomLevelChanged(MapControl sender, object args)
{
    if (MyMap != null)
        mySlider.Value = sender.ZoomLevel;
}

// Locate Me Bottom App Bar
private async void LocateMe_Click(object sender, RoutedEventArgs e)
{
    progressBar.Visibility = Windows.UI.Xaml.Visibility.Visible;
    geolocator = new Geolocator();
    geolocator.DesiredAccuracyInMeters = 50;

    try
    {
        Geoposition geoposition = await geolocator.GetGeopositionAsync(
            maximumAge: TimeSpan.FromMinutes(5),
            timeout: TimeSpan.FromSeconds(10));
        await MyMap.TrySetViewAsync(geoposition.Coordinate.Point, 18D);
        mySlider.Value = MyMap.ZoomLevel;
        progressBar.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
    }
    catch (UnauthorizedAccessException)
    {
        MessageBox("Location service is turned off!");
    }
}

// Custom Message Dialog Box
private async void MessageBox(string message)
{
    var dialog = new MessageDialog(message.ToString());
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, async () => await dialog.ShowAsync());
}

```

Listing 5

Adding Tap Event

One more thing, I want to add is Map Tapped functionality. We will add another method which will give us a cool feature when we tap somewhere and in the Map Icon. It will show the location details in the Message Dialog Box.

To add this method, go to “MainPage.xaml”, select “MapControl” in the main grid. If you are not able to see “Properties” tab, then hit F4 and you’ll find it on the left side of the Visual Studio.

Now double click on the “MapTapped” section and it will automatically generate the method stub for you.

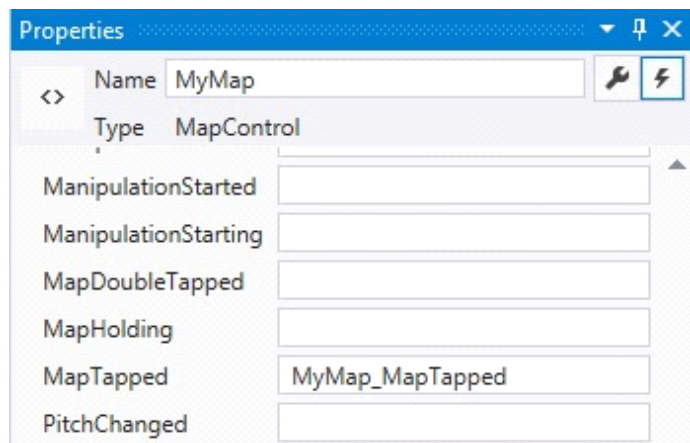


Figure 3

Now, complete the “MyMap_MapTapped” method in the “MainPage.xaml.cs”.

```
private async void MyMap_MapTapped(MapControl sender, MapInputEventArgs args)
{
    Geopoint pointToReverseGeocode = new Geopoint(args.Location.Position);

    // Reverse geocode the specified geographic location.
    MapLocationFinderResult result =
        await MapLocationFinder.FindLocationsAtAsync(pointToReverseGeocode);

    var resultText = new StringBuilder();

    if (result.Status == MapLocationFinderStatus.Success)
    {
        resultText.AppendLine(result.Locations[0].Address.District + ", " + result.Locations[0].Address.Town + ", " +
            result.Locations[0].Address.Country);
    }

    MessageBox(resultText.ToString());
}
```

Listing 6

Adding Location Capability

Now, our work is done, but if you run the application in your device or emulator, you will get an error definitely. Since we have given permission to track our current position to our application, to do this, go to “Package.appxmanifest”, find “Capabilities” section and tick the “Location” service and save it.

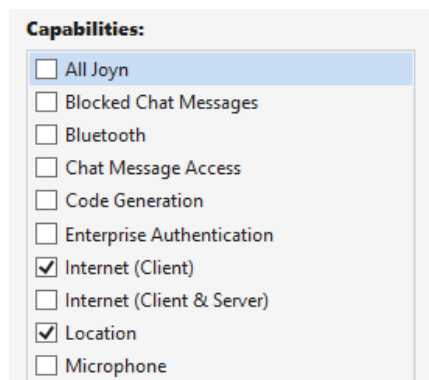


Figure 4

Running the Application

Now if you run the application, it'll look exactly like this.

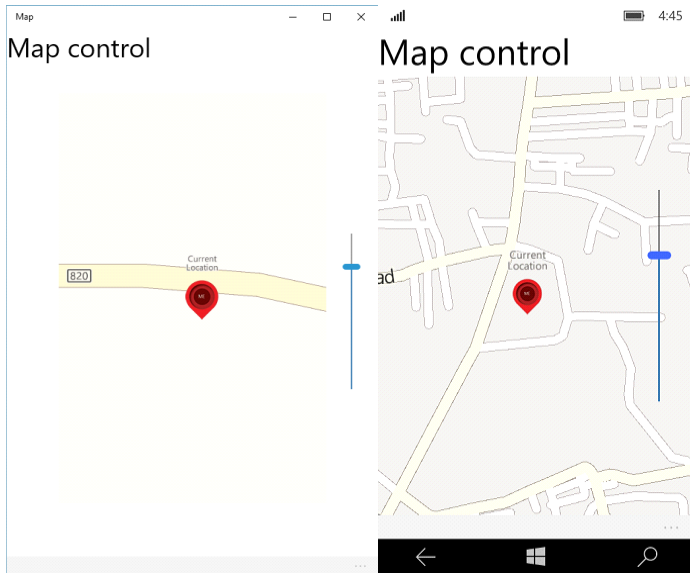


Figure 5

If you tap on the “MapIcon”, it will show the location details on the Message Dialog Box.

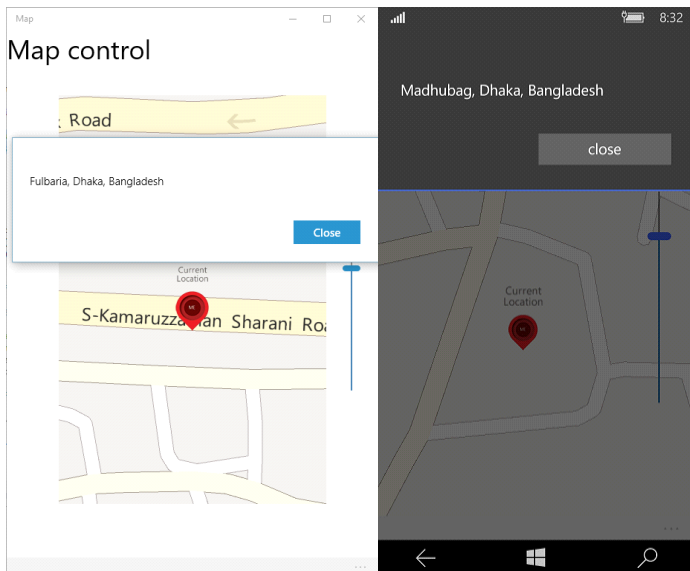


Figure 6

Using Polygon for Pushpin

Another thing, if you don't want to use external image as a “MapIcon”, you can use custom “Pushpin” like “Polygon” control. Then you have to modify the “OnNavigatedTo” method like the one shown below:

```
private async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    // Map Token for testing purpose,
    // otherwise you'll get an alert message in Map Control
    MyMap.MapServiceToken = "abcdef-abcdefghijklmno";

    geolocator = new Geolocator();
    geolocator.DesiredAccuracyInMeters = 50;

    try
    {
        // Getting Current Location
        Geoposition geoposition = await geolocator.GetGeopositionAsync(
            maximumAge: TimeSpan.FromMinutes(5),
```

```

        timeout: TimeSpan.FromSeconds(10));

    // Create a New Pushpin
    var pushpin = CreatePushPin();
    MyMap.Children.Add(pushpin);
    // Setting up Pushpin location
    var location = new Geopoint(new BasicGeoposition()
    {
        Latitude = geoposition.Coordinate.Latitude,
        Longitude = geoposition.Coordinate.Longitude
    });
    MapControl.SetLocation(pushpin, location);
    // Position of the Pushpin
    MapControl.SetNormalizedAnchorPoint(pushpin, new Point(0.0, 1.0));
    // Showing in the Map
    await MyMap.TrySetViewAsync(location, 18D, 0, 0, MapAnimationKind.Bow);

    // Disable the ProgressBar
    progressBar.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
    // Set the Zoom Level of the Slider Control
    mySlider.Value = MyMap.ZoomLevel;
}
catch (UnauthorizedAccessException)
{
    MessageBox("Location service is turned off!");
}
base.OnNavigatedTo(e);
}

```

Listing 7

“CreatePushPin” method is given below.

```

private DependencyObject CreatePushPin()
{
    // Creating a Polygon Marker
    Polygon polygon = new Polygon();
    polygon.Points.Add(new Point(0, 0));
    polygon.Points.Add(new Point(0, 50));
    polygon.Points.Add(new Point(25, 0));
    polygon.Fill = new SolidColorBrush(Colors.Red);

    // Return the Polygon Marker
    return polygon;
}

```

Listing 8

Running the Application

Now, if you run the application, the marker will look like this.

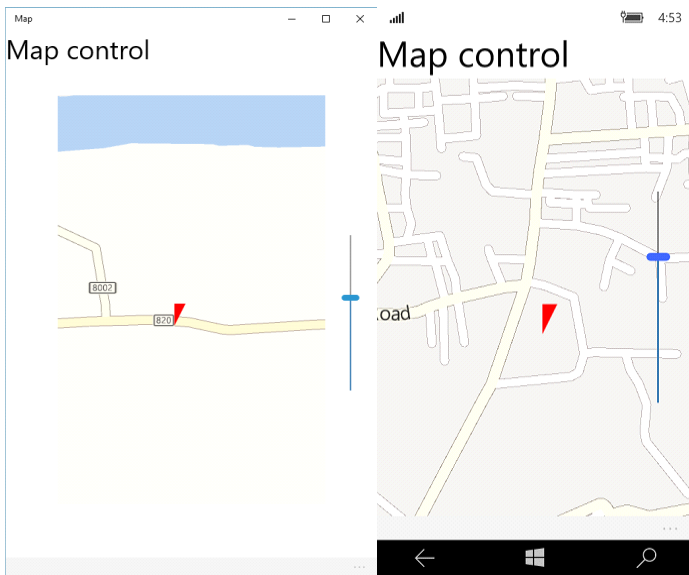


Figure 7

Peach Heading

One more thing I would like to add is Pitch heading. To do this, you need to simply paste the following code in MainPage_Loaded method, under the try-catch block.

```
private async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    ...
    MapScene spaceNeedleScene = MapScene.CreateFromLocationAndRadius(mapIcon.Location,
        400, /* show this many meters around */
        135, /* looking at it to the south east */
        60 /* degrees pitch */);
}
catch()
{
}
}
```

Listing 9

If you run the application, it will give you much different view from the angle of sixty degrees.

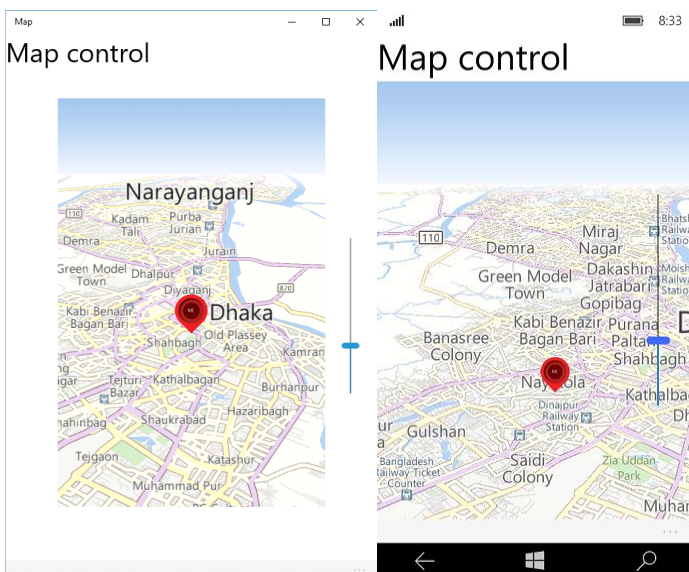


Figure 8

Map Overlaying

One more thing I would like to add is overlaying tiled images on a map. If you want to overlay third-party or custom tiled images on the map and make it customized, then you simply need to use the following lines of code in the “MainPage_Loaded” method at the top.

```
private async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    ...
    var httpsource = new HttpMapTileDataSource("http://a.tile.openstreetmap.org/{zoomlevel}/{x}/{y}.png");
    varts = new MapTileSource(httpsource);
    MyMap.TileSources.Add(ts);
    ...
}
```

Listing: 10

And if you run the application, it will provide you a much more graphical view than before with some details.

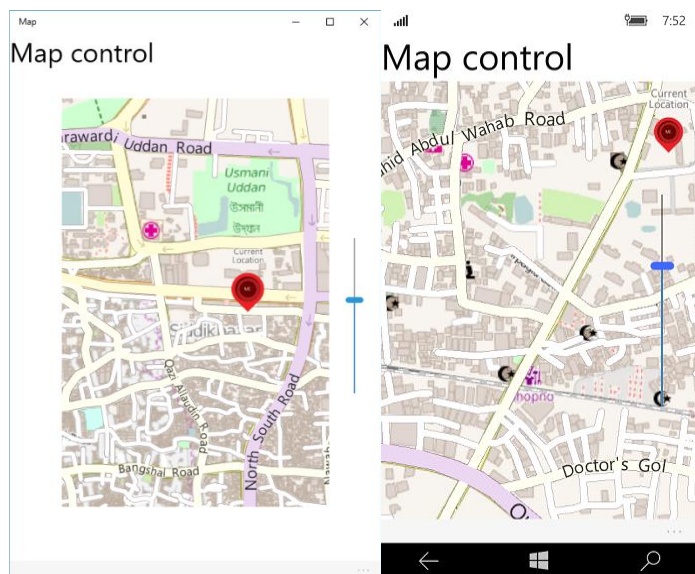


Figure 9

Summary

And that's it! Hope you've understood how to use “MapControl” in Universal Windows Platform. Have fun with Map Control and make some cool applications with it. Read more about Maps and directions at – [GitHub](#).

Happy Coding!

Universal Windows Platform | Azure Mobile Service

Microsoft Azure Mobile Service gives you the power to create a cloud service mobile application. This will make your work more flexible and gives your application more portability in an efficient way. This is a short introduction of Azure Mobile service. You will get to know, how to create and integrate new mobile service in your new or existing application. So let's crack in Azure Mobile Service with Universal Windows Platform.

Create a New Mobile Service

First of all, go to your Azure [portal](#). Select the Mobile Service option and click plus new icon in the bottom of the page.

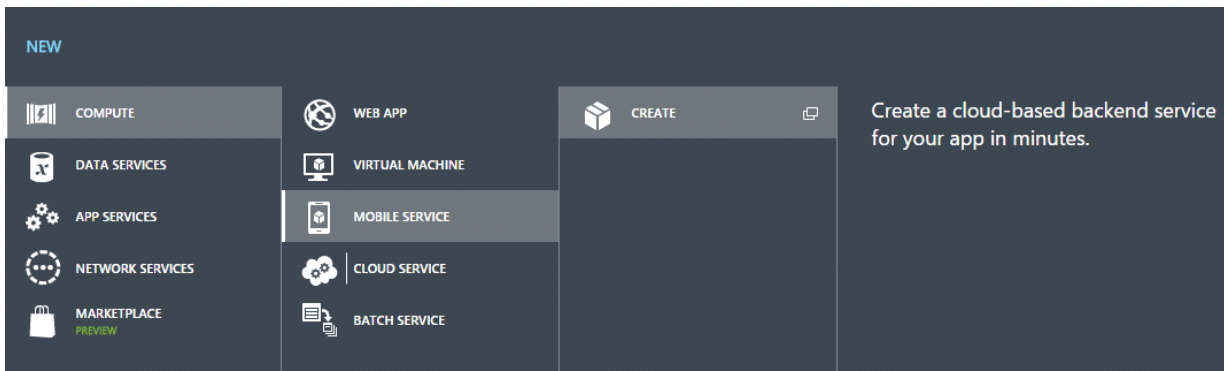


Figure 1: Create a new Mobile Service.

After creating the Mobile Service, open the Mobile Service and select the dropdown option “CONNECT AN EXISTING WINDOWS OR WINDOWS PHONE APP” below the Get Started option.

GET STARTED

- ▶ CREATE A NEW WINDOWS OR WINDOWS PHONE APP
- ▲ CONNECT AN EXISTING WINDOWS OR WINDOWS PHONE APP

Follow these steps to connect an existing application to your mobile service.

1 Connect your app

LANGUAGE:

Right-click your client project, select **Manage NuGet Packages**, search for the `WindowsAzure.MobileServices` package and add a reference to it. Add `using Microsoft.WindowsAzure.MobileServices;`, then copy and paste the following code into your `App.xaml.cs` file:

```
public static MobileServiceClient MobileService = new MobileServiceClient(
    "https://[redacted].azure-mobile.net/",
    "[redacted]");
```

Figure 2: Connection Uri of Mobile Service

Now, copy the code and paste it in the `App.xaml.cs` file of your application. You will get the additional instruction right above the code.

Now create a new class named “Item”, which will make all the necessary tables of your Mobile Service.

```
class Item
{
    [JsonProperty(PropertyName = "id")]
    public string Id { get; set; }

    [JsonProperty(PropertyName = "message")]
    public string Message { get; set; }

    [JsonProperty(PropertyName = "location")]
    public string Location { get; set; }

    [JsonProperty(PropertyName = "Date")]
    public DateTime Date { get; set; }
}
```

Listing: 1

Here we created three attributes or you can say fields `ID`, `Message`, `Location` and `Date`. For `JsonProperty` attribute, you need to install `Newtonsoft.JsonNuGet` package manager. Now, to insert, update and delete the data in your SQL table, we have to write little bit of code in `.cs` files.

Now, design the MainPage.xaml. We need a ListBox to display the data from our Mobile Service table. As there are multiple attributes and columns, here we are showing only Message, Location and Date values.

```
<ScrollViewer>
  <StackPanel Margin="10,10,0,0">
    <ListBoxx.Name="listBox"
      Margin="10,10"
      RequestedTheme="Dark"
      FontSize="20"
      Background="White"
      Foreground="Black" BorderThickness="1"
      BorderBrush="Transparent" >
    <ListBox.ItemContainerStyle>
    <Style TargetType="ListBoxItem">
    <Setter Property="HorizontalContentAlignment" Value="Stretch"></Setter>
    </Style>
    </ListBox.ItemContainerStyle>
    <ListBox.ItemTemplate>
    <DataTemplate>
    <Border BorderThickness="0,0,0,1" BorderBrush="#c0c0c0">
    <StackPanel>
      <TextBlock Foreground="Black" Text="{Binding Message}" />
      <TextBlock Foreground="Black" Text="{Binding Location}" />
      <TextBlock Foreground="Black" Text="{Binding Date}" />
    </StackPanel>
    </Border>
    </DataTemplate>
    </ListBox.ItemTemplate>
    </ListBox>
  </StackPanel>
</ScrollViewer>
```

Listing: 2

Put the code below in MainPage.cs above the constructor.

```
///
```

Listing: 3

Now in constructor, initialize MainPage_Loaded method and implement the code given below.

```
public MainPage()
{
    this.InitializeComponent();
    this.Loaded += MainPage_Loaded;
}

async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    items = await itemTable.ToCollectionAsync();
    this.listBox.ItemsSource = items;
}
```

Listing: 4

Here, 'listbox' is the name of our ListBox control. So, we set item source as 'items', which sets Mobile Service table's values.

If you run the application, it will show all the data in the ListBox control.

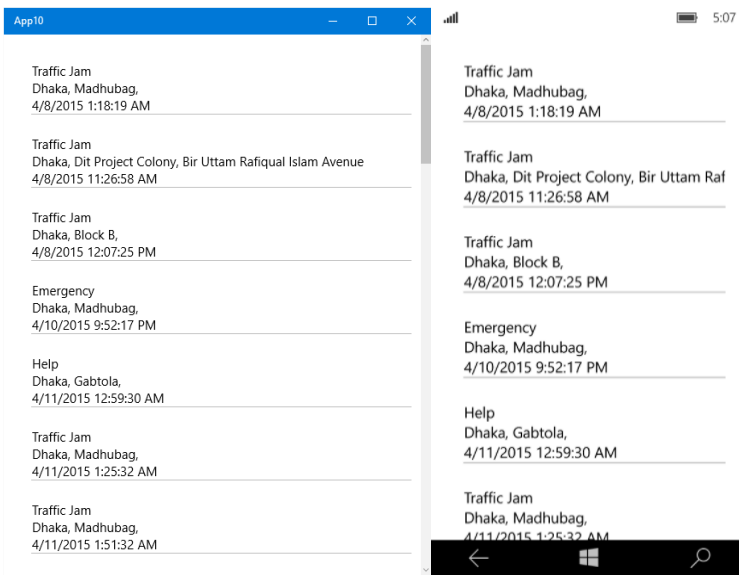


Figure: 3

Hopefully, you have understood the basic of Microsoft Windows Azure Mobile service and the implementation in Universal Windows Platform Application. This is only the basic; you can get deep knowledge by reading the [documentation](#) of Mobile Service and visit the Azure website to get the latest information about Mobile Service. Happy coding!

Universal Windows Platform | Http Client

Web API is the most commonly used cloud service for any type of application. It's flexible and easy to use. It works like a normal http get and post method and other advanced functionality. We will talk about how we can retrieve data and send data to the server via Azure Mobile Service API using [HttpClient](#) NuGet package. So, let's crack in HttpClient in Universal Windows Platform Application.

To make the API we have to go to Azure Mobile Service option. Open your Mobile Service and navigate to the API tab.

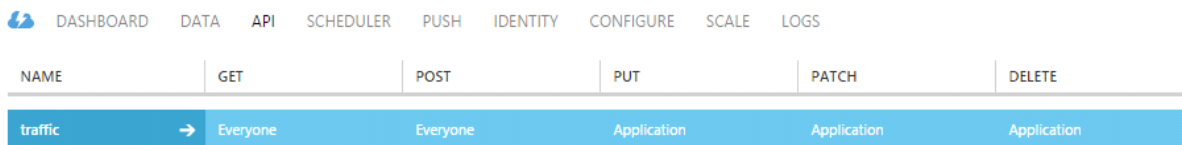


Figure: 1

Now, open the Mobile Service, here we have named it 'traffic' and go to the permission tab. Make the GET & POST permission to Everyone., save it.

api permissions

| | |
|-------------------|----------------------------------|
| GET PERMISSION | Everyone |
| POST PERMISSION | Everyone |
| PUT PERMISSION | Anybody with the Application Key |
| PATCH PERMISSION | Anybody with the Application Key |
| DELETE PERMISSION | Anybody with the Application Key |

Figure: 2

Now move to the Script tab and past the following JavaScript code. Your code may vary due to the Table structure of your Mobile Service.

```

1 exports.post = function(request, response) {
2     // Use "request.service" to access features of your mobile service, e.g.:
3     // var tables = request.service.tables;
4     // var push = request.service.push;
5
6     response.send(statusCodes.OK, { message : 'Hello World!' });
7 };
8
9 exports.get = function(request, response) {
10    var mssql = request.service.mssql;
11    var sql = "select message, location from item"
12    mssql.query(sql,
13    {
14        success:function(results)
15        {
16            response.send(200, results);
17        }
18    })
19 };
20

```

Figure: 3

Here, in the GET method, we make a simple SQL query and send to the client who will request it and in the POST method, we just send the response message 'Hello world', if the request is made successful.

Now, create a new class, give it a name Item, and initialize two properties Message and Location.

```

classItem
{
    [JsonProperty(PropertyName = "message")]
    publicstring Message { get; set; }

    [JsonProperty(PropertyName = "location")]

```

```

    publicstring Location { get; set; }
}

```

Listing: 1

In MainPage.xaml, the design is same as the previous article. We create a simple ListBox control and bind with the Table's attribute.

```

<ScrollView>
    <StackPanel Margin="10,10,0,0">
        <ListBox x:Name="listBox"
            Margin="10,10"
            RequestedTheme="Dark"
            FontSize="20"
            Background="White"
            Foreground="Black" BorderThickness="1"
            BorderBrush="Transparent" >
        <ListBox.ItemContainerStyle>
            <Style TargetType="ListBoxItem">
                <Setter Property="HorizontalContentAlignment" Value="Stretch"></Setter>
            </Style>
        </ListBox.ItemContainerStyle>
        <ListBox.ItemTemplate>
            <DataTemplate>
                <Border BorderThickness="0,0,0,1" BorderBrush="#c0c0c0">
                    <StackPanel>
                        <TextBlock Foreground="Black" Text="{Binding Message}" />
                        <TextBlock Foreground="Black" Text="{Binding Location}" />
                    </StackPanel>
                </Border>
            </DataTemplate>
        </ListBox.ItemTemplate>
        </ListBox>
    </StackPanel>
</ScrollView>

```

Listing: 2

Now in code at the backend, MainPage.xaml.cs file creates a List item named _itemsList of Class Item type and in Constructor generates a Method stub MainPage_Loaded. All the implementation is given below.

```

private List<Item> _itemsList = newList<Item>();

public MainPage()
{
    this.InitializeComponent();
    this.Loaded += MainPage_Loaded;
}

async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        String uri = "_YOUR API URL GOES HERE_";
        JSONArray items = await GetAsync(uri);

        int i = 0;
        int length = items.Count;

        Debug.WriteLine(length);

        while (i < length)
        {
            _itemsList.Add(new Item { Message = items[i]["message"].ToString(), Location =
            items[i]["location"].ToString() });
            Debug.WriteLine(items[i]["message"].ToString() + " " + items[i]["location"].ToString()); // For testing
            purpose.
            i++;
        }
    }
}

```

```

this.listBox.ItemsSource = _itemsList;

Item item = newItem
{
    Message = "",
    Location = ""
};

string jsonData = JsonConvert.SerializeObject(item);

JsonObject message = await PostAsync(uri, jsonData);
Debug.WriteLine(message); // For testing purpose.

}
catch (Exception ex)
{
    Debug.WriteLine(ex.Message);
}
}

```

Listing: 3

Here, first we need the API URL, Json Array (JArray) and we get it from the given URL. We get the length of the Json array and a loop while it becomes zero. Inside the loop, we insert the items in to the List that we have created before and then make it the ItemSource of the ListBox.

Subsequently we have created a new instance of Item class and declare two null values for the POST operation. After that, we send the data to the server by serializing it as a Json format. The GetAsync and PostAsync methods are given below.

```

private async Task<JArray> GetAsync(string uri)
{
    var httpClient = new HttpClient();
    var content = await httpClient.GetStringAsync(uri);
    return await Task.Run(() => JArray.Parse(content));
}

public async Task<JsonObject> PostAsync(string uri, string data)
{
    var httpClient = new HttpClient();
    var response = await httpClient.PostAsync(uri, new StringContent(data));

    response.EnsureSuccessStatusCode();

    string content = await response.Content.ReadAsStringAsync();
    return await Task.Run(() => JsonObject.Parse(content));
}

```

Listing: 4

The GetAsync will return the Json Array from the Uri and the PostAsync will return the success message if the data transfer is done successfully. It will check if there is any error by response or not.

If we run the Application, it will give the same result but an attribute less than the previous article.

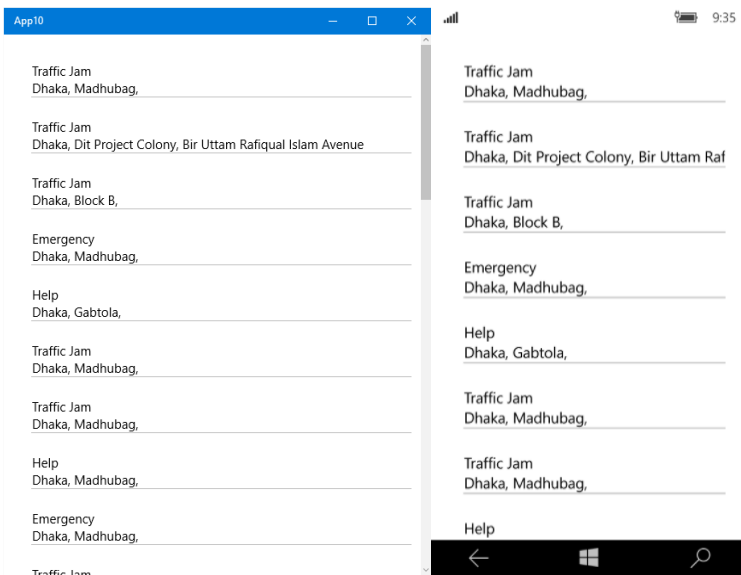


Figure: 4

To test your API URL, you can use Postman Google Chrome App. This is a good way to test the URL is working or not.

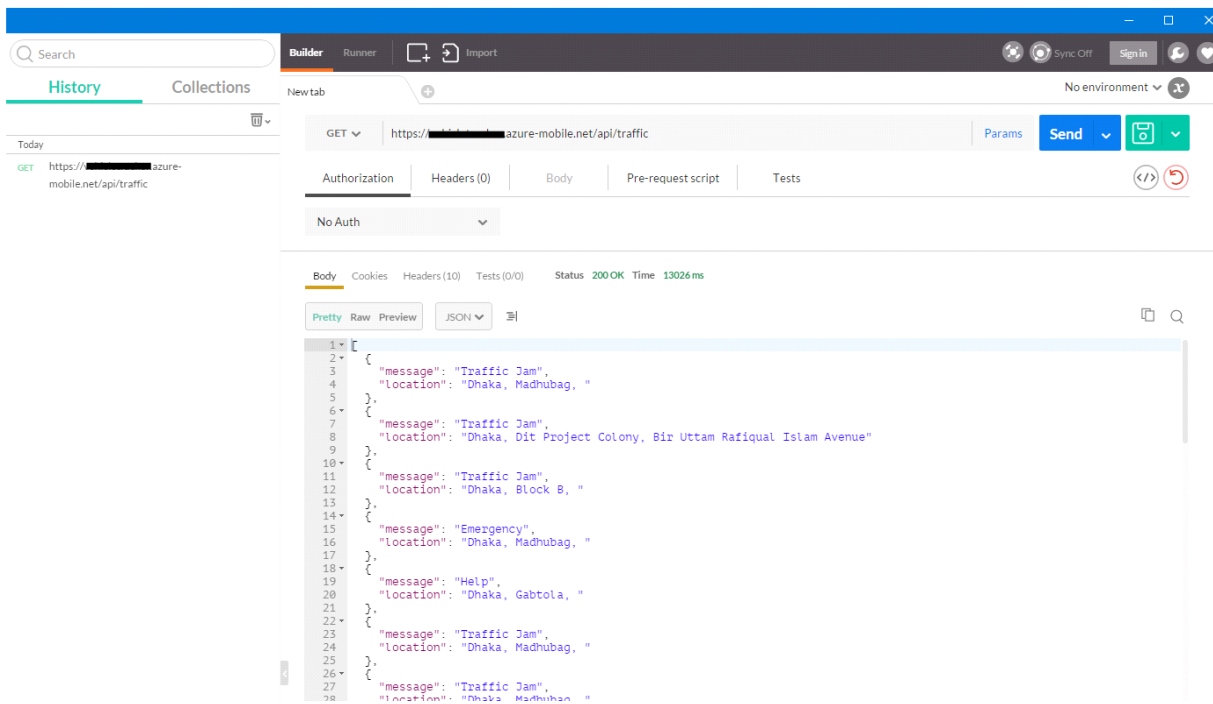


Figure: 5

You can view the data in different format. One can test the URL by GET, POST and many other http methods.

If you want to see the POST is successful or not, you can see the Output windows. If you see the success message that you have sent from the server, then you are good to go.

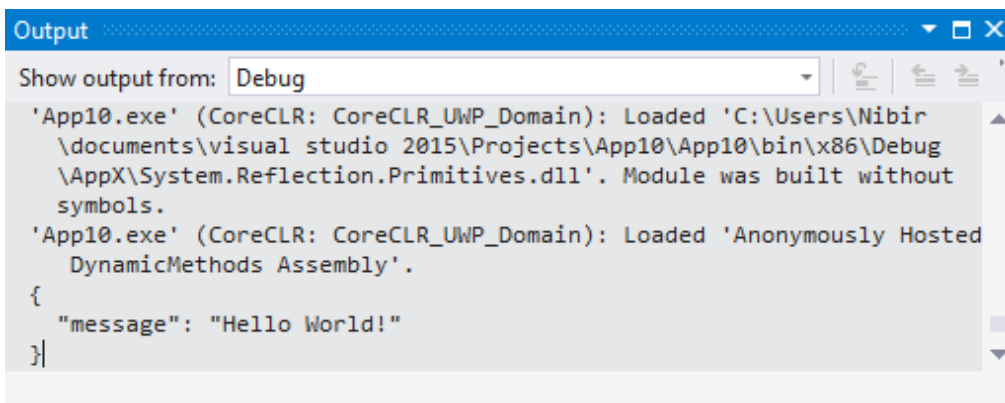
The image shows a screenshot of the Visual Studio Output window. The window title is 'Output'. Below the title bar, there is a dropdown menu set to 'Debug'. The output text shows two messages from 'App10.exe' (CoreCLR: CoreCLR_UWP_Domain). The first message reports the loading of 'C:\Users\Nibir\documents\visual studio 2015\Projects\App10\App10\bin\x86\Debug\AppX\System.Reflection.Primitives.dll' and notes it was built without symbols. The second message reports the loading of 'Anonymously Hosted DynamicMethods Assembly'. Below these messages, a JSON object is displayed: { "message": "Hello World!" }.

Figure: 6

This is the same message in Figure 3 which we we have set in line number 6.

This article is for a demonstration purpose and tells you that how you can use the Web API in your Universal Windows Platform Application. Hope this will help to understand the ins and outs of HttpClient Nuget package and handling Json data. Happy coding!

Universal Windows Platform | Live tiles & Push Notification

Live tiles and push notification are the essential features of Universal Windows Platform Application. These are the key features of Modern Application. These show important information without opening the application which runs in the background. So, it's really helpful for getting the required information just looking at the phone to get notified when new messages or updates arrives.

Live tiles

Live tiles usually show some information of the application just glancing at the screen. It updates automatically. In Universal Windows Platform Application, there are four types of tiles available (e.g., small, medium, wide and large).

To implement Live tiles in your application is really an easy task. First, create a simple button control.

```
<StackPanel>
    <Buttonx:Name="button" Height="Auto" Width="Auto" Content="Notification Agent" Click="button_Click"
        Margin="10,10,0,0"/>
</StackPanel>
```

Listing: 1

For Live tiles, we need to create a new instance of SecondaryTile and we want to show the current date-time on that. So, the code that is functioning at the backend is given below.

```
var tileID = "DateTile";

SecondaryTile tile = new SecondaryTile(tileID, DateTime.Now.ToString(), "args", new Uri("ms-appx:///Assets/DefaultSecondaryTileAssests/Medium.png"), TileSize.Default);
tile.VisualElements.Square71x71Logo = new Uri("ms-appx:///Assets/DefaultSecondaryTileAssests/Small.png");
tile.VisualElements.Wide310x150Logo = new Uri("ms-appx:///Assets/DefaultSecondaryTileAssests/Wide.png");
tile.VisualElements.Square310x310Logo = new Uri("ms-appx:///Assets/DefaultSecondaryTileAssests/Large.png");
tile.VisualElements.ShowNameOnSquare150x150Logo = true;
tile.VisualElements.ShowNameOnSquare310x310Logo = true;
tile.VisualElements.ShowNameOnWide310x150Logo = true;

await tile.RequestCreateAsync();
```

Listing: 2

Here, we have created a tileID which is unique for any tiles. Afterwards, we create a new instance of SecondaryTile and pass the tileID, current date-time, set the medium tile image for that. Next, we've set three different sizes of the square logos for that and made their visibility to true. Finally, we have set the RequestCreateAsync for the specific tile. Most importantly, the button click method is not async initially, so we modified it as an async method because the RequestCreateAsync method is an await operation. Now, if you run the application, after which button is clicked? A new window will pop up because we have set that in the Secondary tile constructor. Click yes and the tile will appear in your start menu.

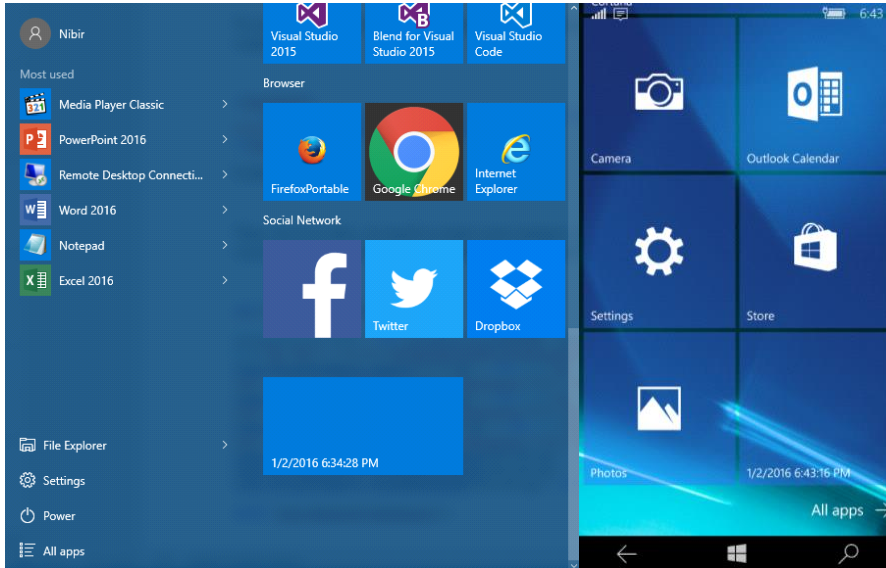


Figure: 1

Toast Notification

A Toast Notification is a window element which displays some message or information for Universal Windows Platform Application. It will also navigate to the related window of the notified segment. Similar to the Live tiles, just make a button control (see listing 1) or you can use the same code block for this.

The implementation is quite easy. It has mainly three steps, firstly make the template, secondly put the information you want to display and lastly the toast.

```
var template = ToastTemplateType.ToastText01;
var xml = ToastNotificationManager.GetTemplateContent(template);
xml.DocumentElement.SetAttribute("launch", "Args");

var text = xml.CreateTextNode(DateTime.Now.ToString());
var elements = xml.GetElementsByTagName("text");
elements[0].AppendChild(text);

var toast = new ToastNotification(xml);
var notifier = ToastNotificationManager.CreateToastNotifier();
notifier.Show(toast);
```

Listing: 3

So, we made a single text template and define the template like a XML template. We set the text of today's date, time and pass the XML template to the toast notification.

If you run the application, it will look like this.

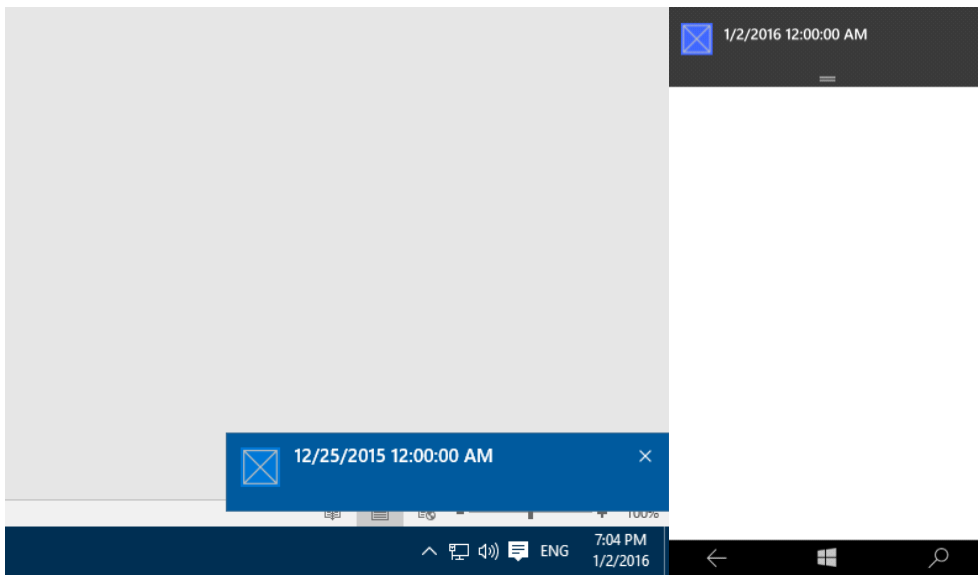


Figure: 2

You can also find it in the Action Center.

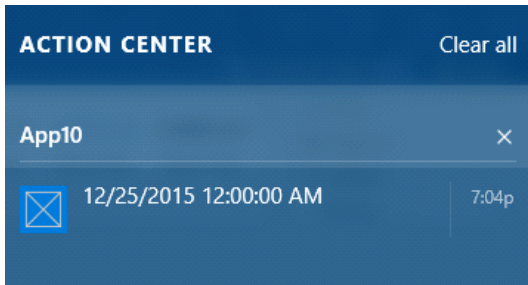


Figure: 4

Hopefully you understood the procedure and implementation of Live tiles and Toast Notification agent. These are really helpful and make your application more complete and professional. Happy coding!

Universal Windows Platform | Local Storage

Local Storage is really helpful to store small amount of data. If you want to save some settings or something, you can use local storage rather than other storage services. It's easy to use and flexible of course. So, let's see how we can use it in our Universal Windows Platform Application.

To start with Local Storage, we will show a really simple example of this. For basic demonstration purpose, in MainPage.xaml we have taken two TextBlock wrapped with a StackPanel. First TextBlock will show the application starting date and the second one will show the time left of a seven days' time trial.

```
<StackPanel>
  <TextBlock Name="textBlock"FontSize="36" Height="50" Width="Auto" Margin="10,10,0,0" Text="Starting date."/>
  <TextBlock Name="textBlock1"FontSize="36" Height="50" Width="Auto" Margin="10,10,0,0" Text="Time left."/>
</StackPanel>
```

Listing: 1

MainPage.xaml.cs defines a OnNavigatedTo method and puts the relative code inside the block.

```
protectedoverrideasyncvoidOnNavigatedTo(NavigationEventArgs e)
```

```

{
    Windows.Storage.ApplicationDataContainer localSettings = Windows.Storage.ApplicationData.Current.LocalSettings;

    if ((string)localSettings.Values["IsFirstTimeLaunched"] == "true")
    {
        textBlock.Text = localSettings.Values["date"].ToString();
    }
    else
    {
        DateTime start = DateTime.Now;

        localSettings.Values["IsFirstTimeLaunched"] = "true";
        localSettings.Values["date"] = start.Date.ToString();

    }

    DateTime start1 = DateTime.Parse(localSettings.Values["date"].ToString());
    DateTime end = DateTime.Now;
    int trialPeriod = 7;

    if ((end.Date - start1.Date).Days <= trialPeriod)
    {
        textBlock1.Text = "You have " + (trialPeriod - (end.Date - start1.Date).Days).ToString() + " left";
    }
}

```

Listing: 2

First we have created ApplicationDataContainerlocalsettings and check a setting “IsFirstTimeLaunched”. If it’s true, we show the value otherwise set it “true”. We set the current date-time to it. So, it will set only once because only the first time, the if statement will be false and for the rest of the time it will be always true.

Now, we want to set a seven days’ time period, so that we can count the time elapsed from the start. We create a local variable “end” and “trialPeriod”. We check if the difference between start date and end date is less than seven then we display the time left in the second TextBlock.

If you run the application, it will look like this.

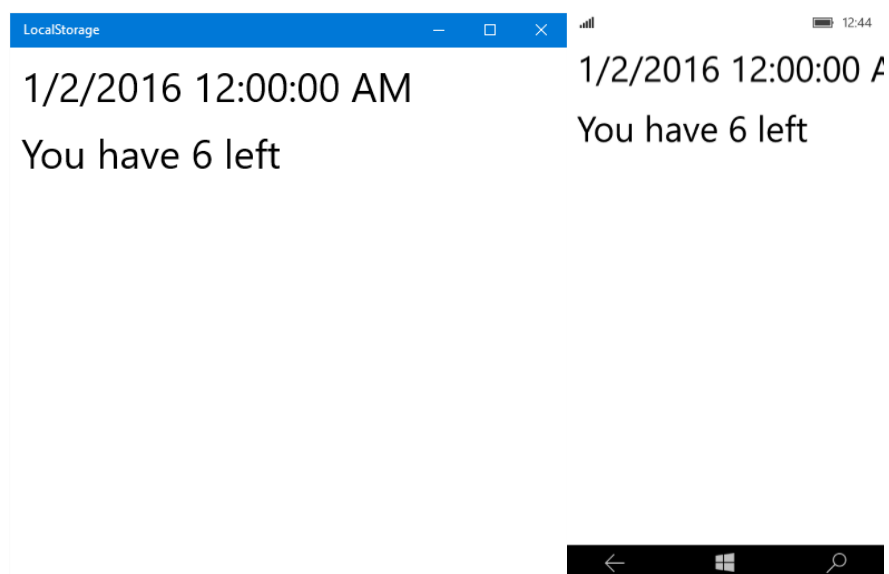


Figure: 1

So, this is a short description of using local storage in Universal Windows Platform Application. Hope this will help.

Barcode Scanner Cross Platform App Using Cordova in Visual Studio 2015

Cordova is an open source framework for quickly building cross-platform mobile apps using HTML5, JavaScript, CSS etc.

Cordova is an application container technology that allows you to create natively-installed applications for mobile devices using HTML, CSS and JavaScript. The core engine for Cordova is also 100% open source, under the Apache Cordova project.

The UI layer of a Cordova application is a web browser view that takes up 100% of the device width and 100% of the device height.

Cordova provides an application programming interface (API) that enables you to access native operating system functionality using JavaScript. You build your application logic using JavaScript and the PhoneGap API handles communication with the native operating system.

Cordova currently supports all major mobile platforms:

- iOS
- Android
- Windows Phone 8

In this article we will see the step by step setup Visual Studio 2015 setup process for developing Apache/Cordova cross platform mobile application development and then develop a barcode scanner app using phonegap barcode scanner plugin.

At the time of installing Visual Studio 2015, you need to select Cross Platform Mobile Development from the features section and then select HTML5 / JavaScript (Apache Cordova) Update 2 /3 from the sub section. Click install button to complete the installation process.

To see full installation process, please follow this article:

<http://www.c-sharpcorner.com/UploadFile/020f8f/visual-studio-enterprise-2015-ide-installation-process/>

Here, I am using Visual Studio Enterprise 2015 version. You can use any other version too like Visual Studio Professional 2015.

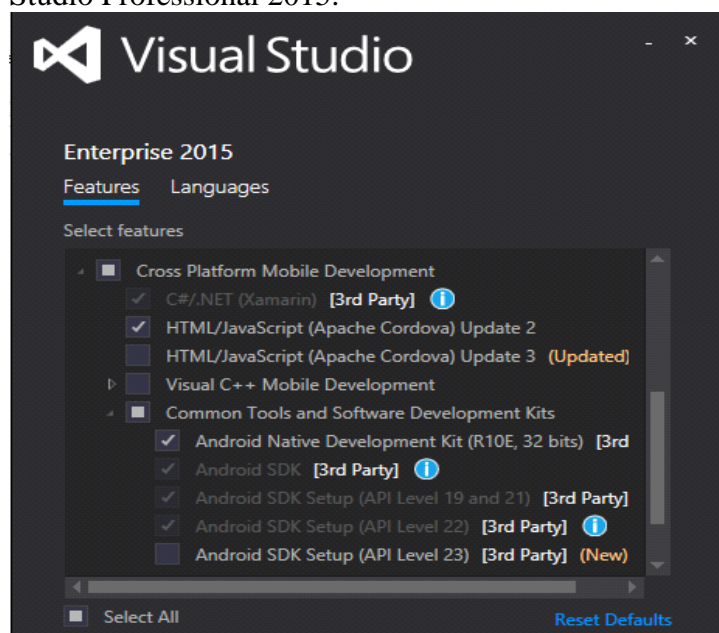


Figure: 1

After completing the installation process, please open visual studio 2015.

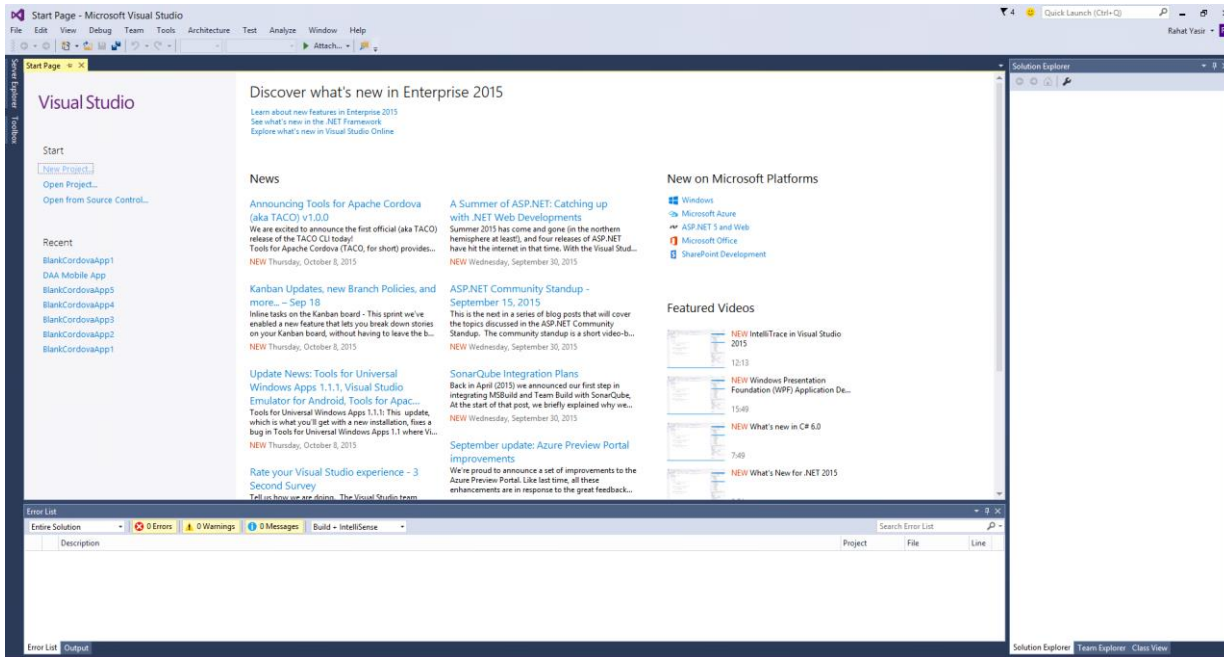


Figure: 2

Click on the File menu and subsequently click on the New menu. Select Project from there. Then you will see the project template menu like below figure. Select JavaScript and click on the Apache Cordova Apps section. Then select Blank App (Apache Cordova from the app template section. Now write a name of the project and then click OK button.

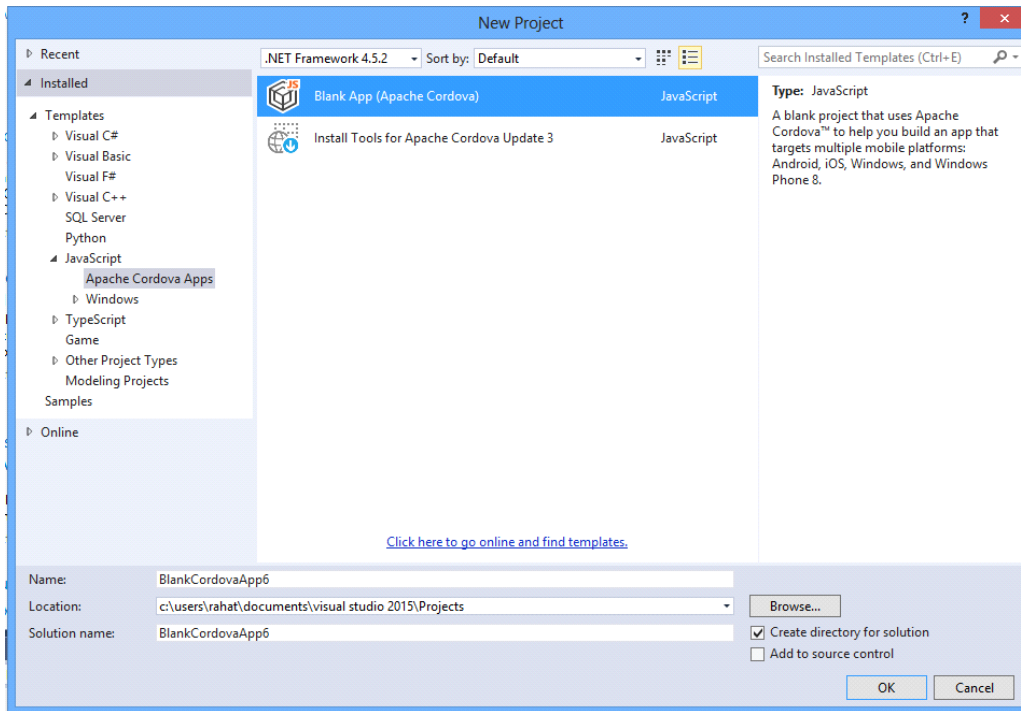


Figure: 3

Now a new Cordova project will be created and then you will see newly created Cordova files in the Solution explorer. Here in the merges folder, you will see different platform related JavaScript libraries. In the dependencies section, you will see Cordova client side package manager related files like npm and bower files. If you install any Phonegap plugin then a new plugin folder will appear and plugin codes will be on that folder. On the res folder, you will see resource files like icons for different platforms, android ant property files and platform based screen images. On the www folder, we will see our HTML5, css and javascript files. We will mostly write our codes here. On the config.xml file, you can change configuration of your project, add or remove plugins, set names and versions for different platforms.

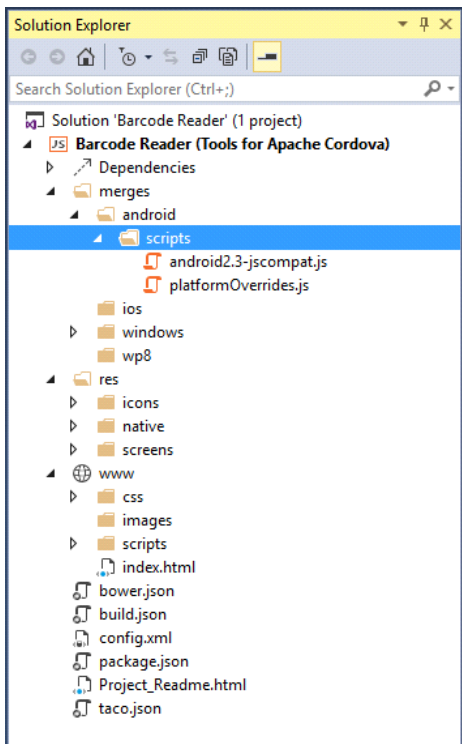


Figure: 4

Now click on the index.html page. Then you will see below codes for index.html.

```

<!DOCTYPEhtml>
<html>
  <head>
    <metacharset="utf-8"/>

    <!--
      Customize the content security policy in the meta tag below as needed. Add 'unsafe-inline' to default-src
      to enable inline JavaScript.
      For details, see http://go.microsoft.com/fwlink/?LinkID=617521
    -->
    <metahttp-equiv="Content-Security-Policy"content="default-src 'self' data: gap: https://ssl.gstatic.com 'unsafe-
    eval'; style-src 'self' 'unsafe-inline'; media-src *">
    <title>BarcodeReader</title>

    <!-- BarcodeReader references -->
    <linkhref="css/index.css"rel="stylesheet"/>
  </head>
  <body>
    <p>Hello, your application is ready!</p>

    <!-- Cordova reference, this is added to your app when it's built. -->
    <scriptsrc="cordova.js"></script>
    <scriptsrc="scripts/platformOverrides.js"></script>

    <scriptsrc="scripts/index.js"></script>
  </body>
</html>

```

Listing: 1

In the above code, src="cordova.js" is referencing updated installed Cordova library and src="scripts/platformOverrides.js" file is referencing the library platform independent of this application.

Since we already created a Cordova cross platform app, we will create a barcode scanner app. Click on the confix.xml file. You will see below screen.

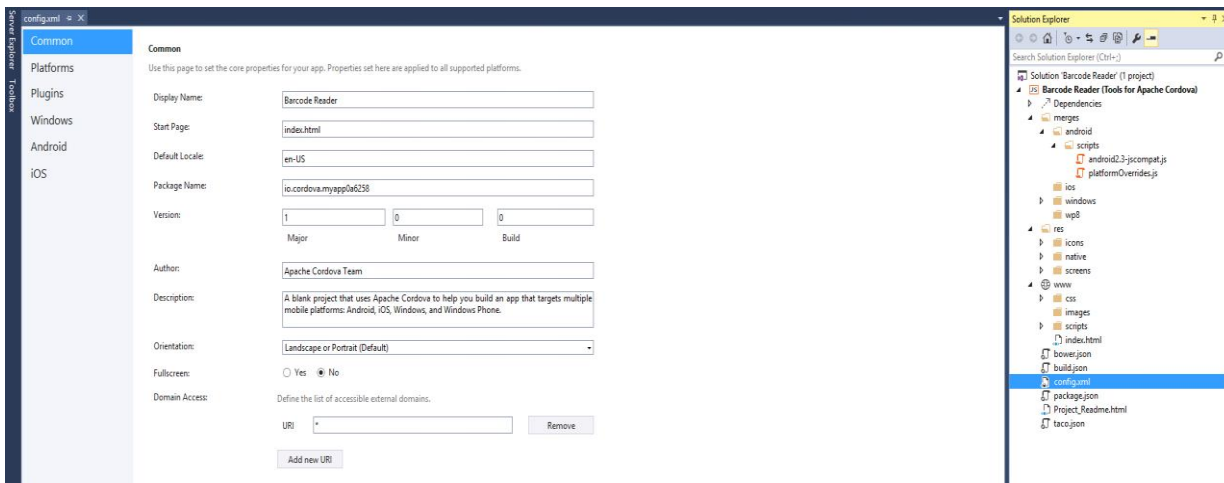


Figure: 5

Now, if you click on the Platforms section, then you will see current installed Cordova version of this project. Now click on the Plugins section and then you will see a list of core plugins like below screen. You can add any plugin by clicking on the “Add” button.

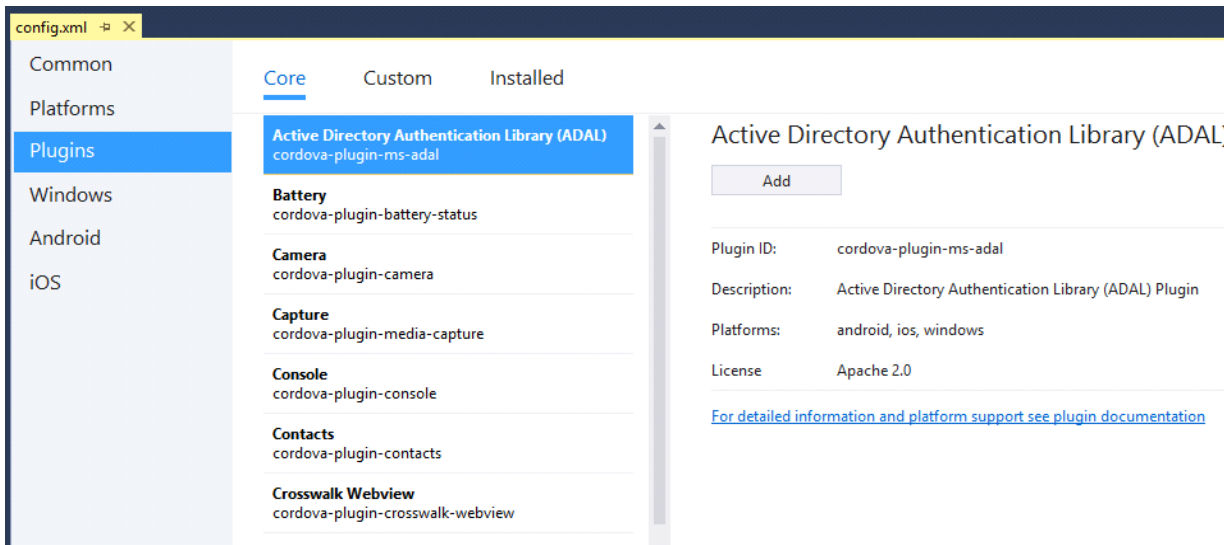


Figure: 6

There is a cross platform Barcode Scanner library for Cordova and PhoneGap platform in Git. Please follow this link to see the Git repository of that Barcode Scanner plugin.

Link: <https://github.com/phonegap/phonegap-plugin-barcodescanner>

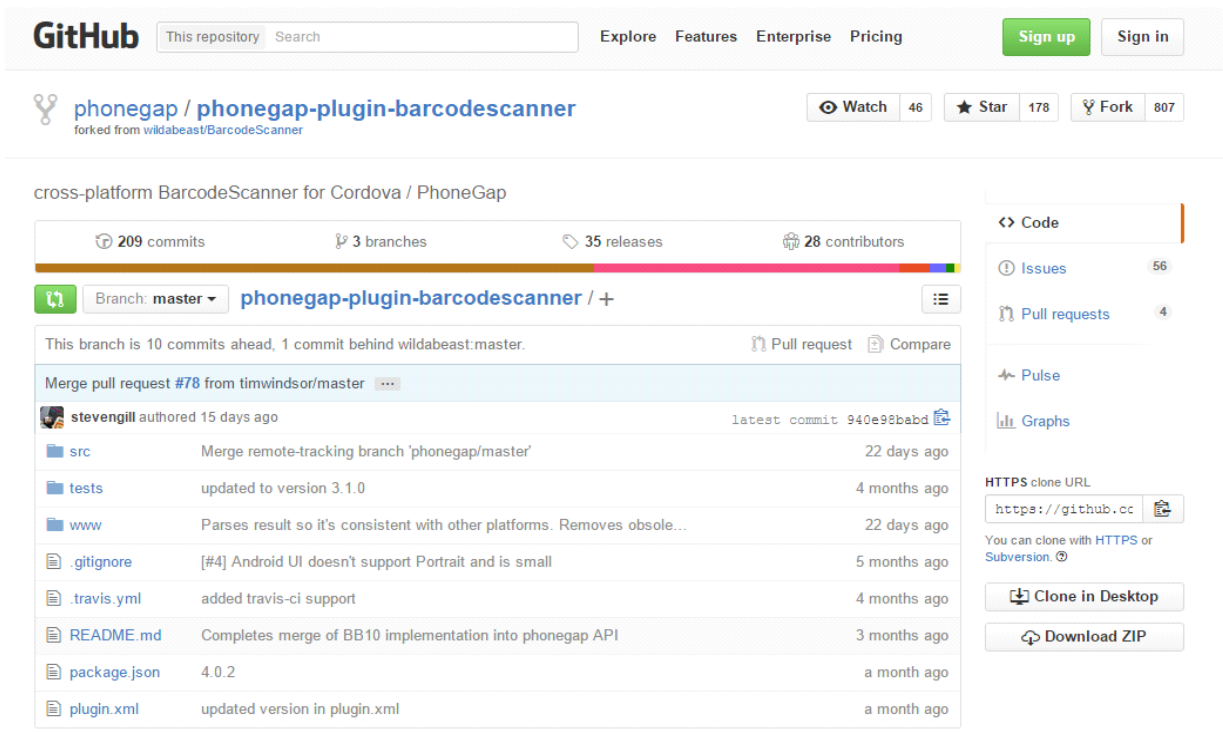


Figure: 7

Now we will add that plugin in our application. Select Custom section from the Plugins like below. Select Git radio button and paste the Git repository link. Then click on the arrow button on the left.

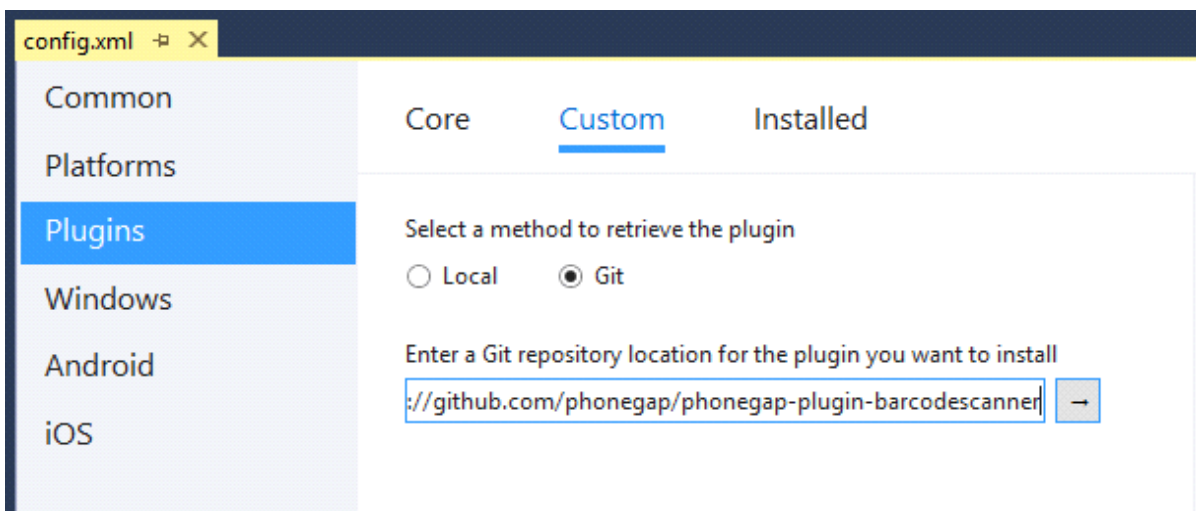


Figure: 8

Then you will see the Barcode Scanner plugin on the right side of the config.xml page. Click “Add” button to add this plugin in your Cordova app.

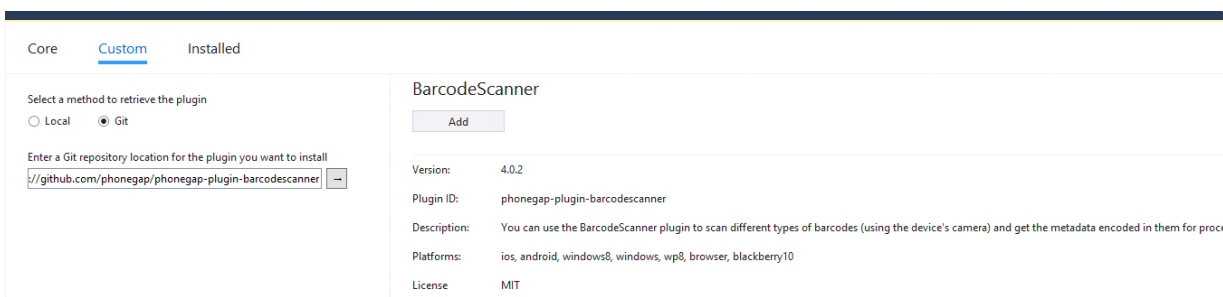


Figure: 9

After few seconds, you will see this message “This plugin was successfully installed”. If you want to remove the plugin then you just need to click on the remove button but don’t remove it now. We will use this plugin to develop our Barcode Scanner app.

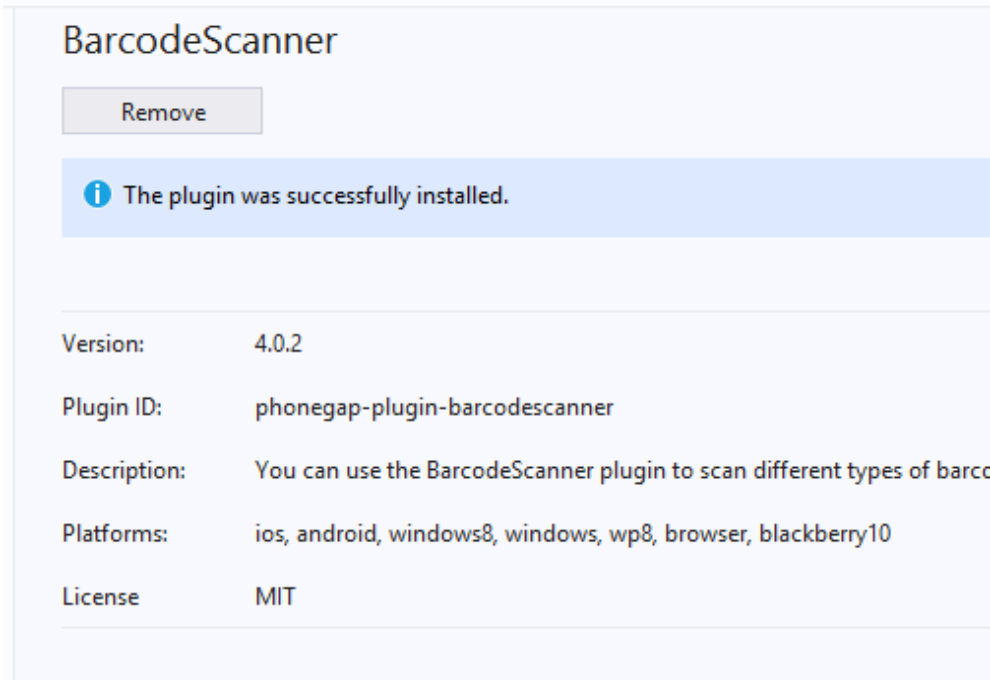


Figure: 10

Now click on the Installed tab on the Plugins section and you will see that BarcodeScanner plugin is already installed in your project.

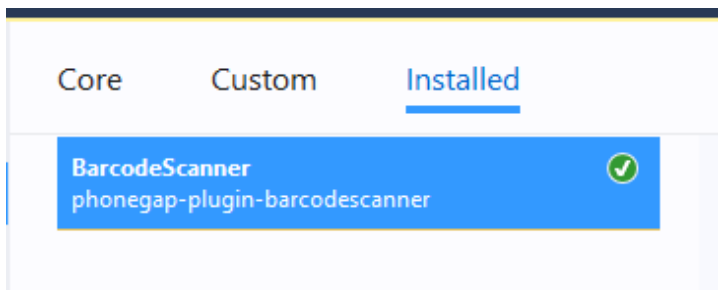


Figure: 11

As we use our mobile phone camera to capture the barcode and scan it, add two more plugins from the Core plugins tab. Install Camera plugin and Capture plugin from the Core plugins section.

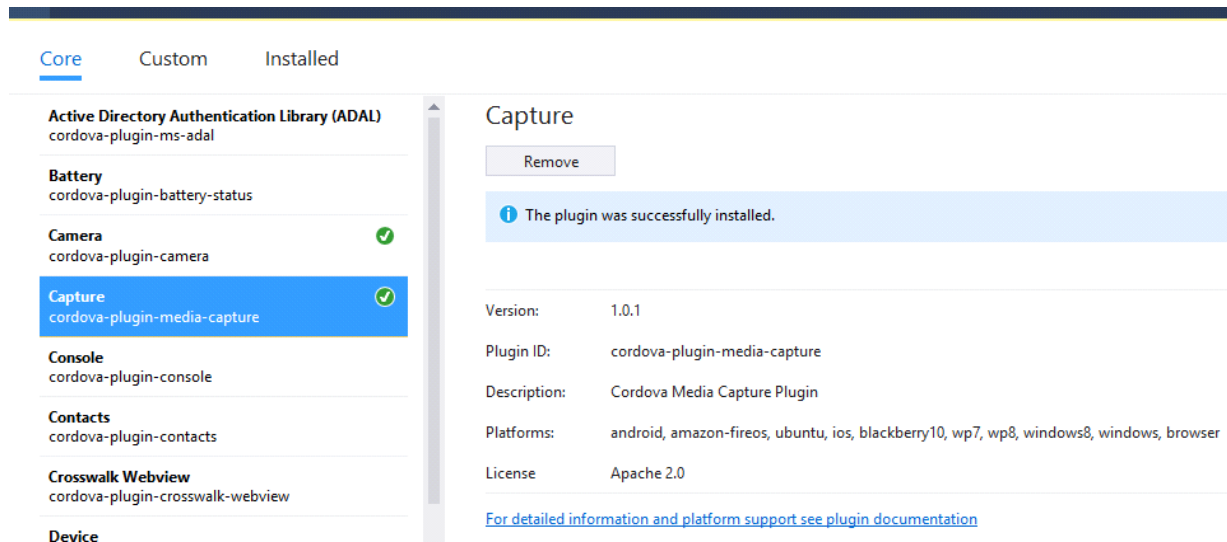


Figure: 12

Now add a button in the body section of the HTML page like the code given below.

```
<buttonid="scan"onclick="scan()"style="width:100px; height: 30px;">Scan</button>
```

Listing: 2

Now add below JavaScript function code with in the head tag in the script section. Here, we are calling the barcode scanner plugin through *cordova.plugins.barcodeScanner.scan* API and when we call this scan section, a default camera capture task will appear on the screen and then, the user will scan the barcode. If our app successfully finds a barcode within the frame of camera capture then the result will be true, we will get barcode value and other information through a popup window.

```
function scan() {
    cordova.plugins.barcodeScanner.scan(
function (result) {
    alert("We got a barcode\n" +
        "Result: " + result.text + "\n" +
        "Format: " + result.format + "\n" +
        "Cancelled: " + result.cancelled);
    },
function (error) {
    alert("Scanning failed: " + error);
    }
    );
}
```

Listing: 3

Here is the full code of index.html page.

```
<!DOCTYPEhtml>
<html>
  <head>
    <metacharset="utf-8"/>

    <!--
      Customize the content security policy in the meta tag below as needed. Add 'unsafe-inline' to default-src to enable
      inline JavaScript.
      For details, see http://go.microsoft.com/fwlink/?LinkID=617521
    -->
    <metahttp-equiv="Content-Security-Policy"content="default-src 'self' data: gap: https://ssl.gstatic.com 'unsafe-eval';
    style-src 'self' 'unsafe-inline'; media-src *">
    <title>BarcodeReader</title>

    <!-- BarcodeReader references -->
```

```

<linkhref="css/index.css"rel="stylesheet"/>
<script>
    function scan() {
        cordova.plugins.barcodeScanner.scan(
    function (result) {
        alert("We got a barcode\n" +
            "Result: " + result.text + "\n" +
            "Format: " + result.format + "\n" +
            "Cancelled: " + result.cancelled);
    },
    function (error) {
        alert("Scanning failed: " + error);
    }
    );
}
</script>
</head>
    <body>
    <p>Hello, your application is ready!</p>
    <buttonid="scan"onclick="scan()"style="width:100px; height: 30px;">Scan</button>
<!-- Cordova reference, this is added to your app when it's built. -->
<scriptsrc="cordova.js"></script>
<scriptsrc="scripts/platformOverrides.js"></script>

<scriptsrc="scripts/index.js"></script>
</body>
</html>

```

Listing: 4

Now build this solution and you will see that build is successful and no error found.

```

100 %
Output
Show output from: Build
1> Writing out cordova_plugins.js...
1> Install complete for phonegap-plugin-barcodeScanner on android.
1> saving android@4.0.2 into platforms.json
1> ----- Updating plugins
1> ----- Currently installed plugins: cordova-plugin-whitelist@1.0.0, phonegap-plugin-barcodeScanner@4.0.2
1> ----- Currently installed dependent plugins:
1> ----- Currently configured plugins: phonegap-plugin-barcodeScanner@4.0.2
==== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====

```

Figure: 13

If you are not using updated npm and node js version, you may see few errors and build failed message. Then, update node js and npm and then, you will see that the build is succeeded.

You can select any platform from the combo box besides debug. You can select Android, iOS, Windows Phone etc. You can build Windows phone and Android cross platform apps through this process but to build your app for iOS platform, you need to connect your visual studio with a mac machine. Here, I have selected Android platform and connected my android device through USB cable with my pc.

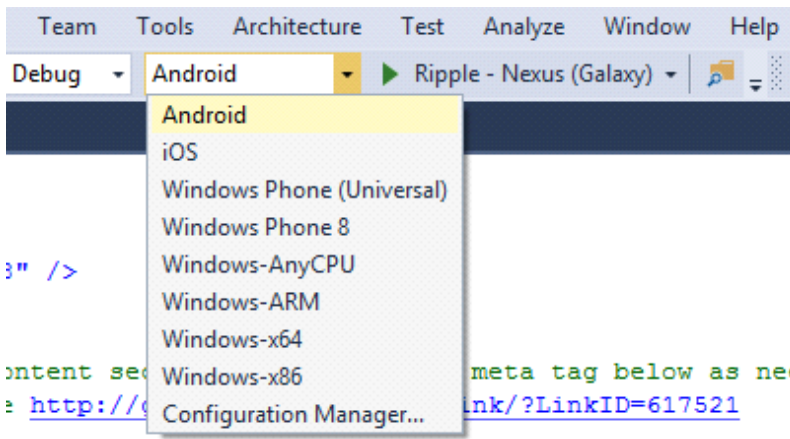


Figure: 14

Now select the Device type from the device section just besides the platform section. You can deploy your cross platform app in your real android device if you select Device from the combo box. You can also select Nexus 7, Nexus S or, Nexus Galaxy Ripple emulators to run your developed apps.

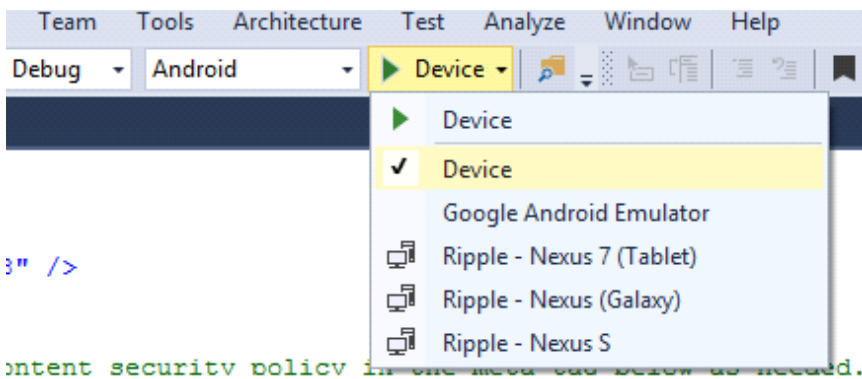


Figure: 15

After deploying this Barcode Scanner app on my android device, I will see below screen on my mobile device.

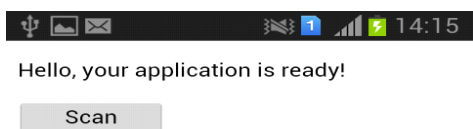


Figure: 16

Now click on the scan button, and then you will see below camera capture task screen. Now, take your mobile device and scan any barcode you want.

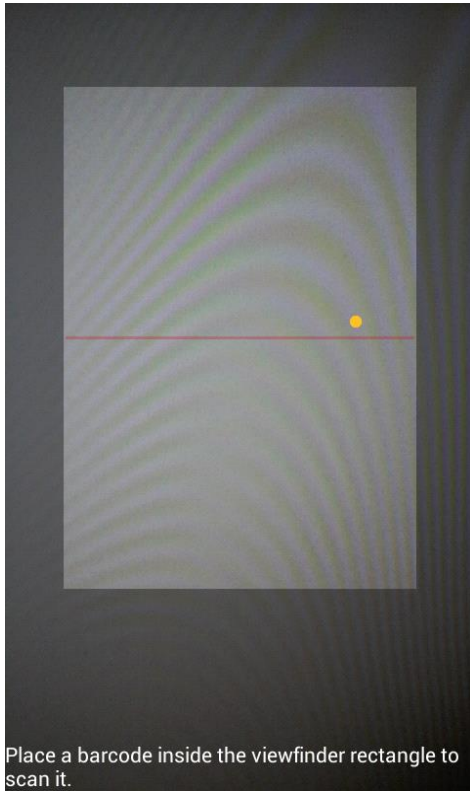


Figure: 17

I will scan below barcode to get the barcode value of this barcode by scanning it using my newly developed barcode scanner app.



Figure: 18

Below, you can see that I am capturing the above barcode image through my mobile device app. "Place a barcode inside the viewfinder rectangle to scan it".

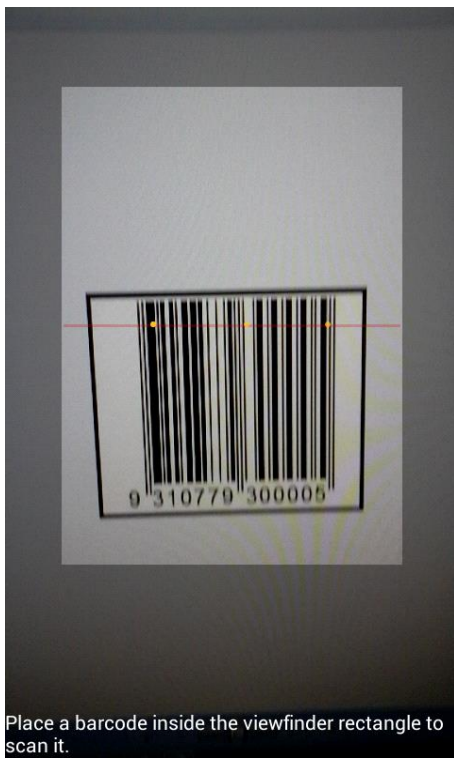


Figure: 19

Below, you can see that our barcode scanner apps have successfully found a barcode in the camera capture frame. Whenever our app will find a barcode, the texts at the bottom of the screen will change from “Place a barcode inside the viewfinder rectangle to scan it” to “Found Product”.

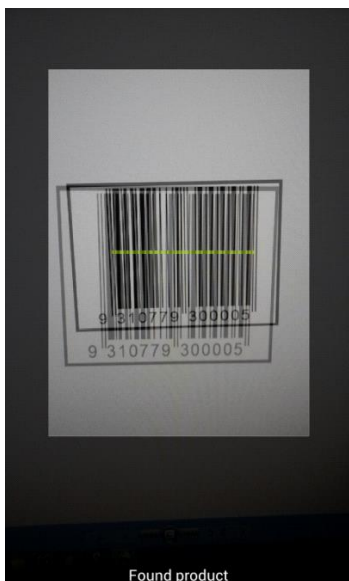


Figure: 20

Afterwards you will see a popup message with a message like “We got a barcode”, barcode value and Format of the barcode. Yes, that’s it. We have easily developed a Barcode Scanner app in just few minutes. Congrats.

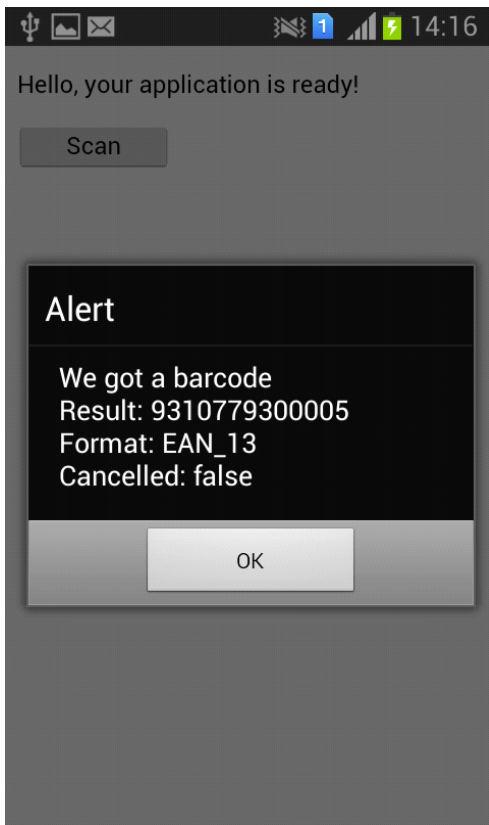


Figure: 21

Hopefully, this helps you to develop powerful Cross Platform Application with Visual Studio 2015. Visual Studio 2015 support different kinds of OS targeted application with great performance. So, make amazing Cross Platform Application and cut down your work stretch unlike before. Happy coding!

Appendix

All the chapters are blogged in our blog site.

If you find any difficulties and have any question, you can ask there. <http://learnwithbddevs.wordpress.com/>

You can read articles of this book from this site,

<http://www.c-sharpcorner.com/Authors/020f8f/Articles/>

All the chapters' source code can be [found here](#),