LEARNING uwp

#uwp

Table of Contents

About

Chapter 1: Getting started with uwp

Remarks

This section provides an overview of what uwp is, and why a developer might want to use it.

It should also mention any large subjects within uwp, and link out to the related topics. Since the Documentation for uwp is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting UWP set up or installed.

Requirements

1. Windows 10
2. Visual Studio 2015

Steps

- Download and custom install Visual Studio 2015, while making sure that `Universal Windows App Development Tools`is selected along with its sub options:-
  a) *Tools and Windows SDK*
  b) *Emulator for Windows Phone*

- Make sure to Enable Developer Mode on development and deploying device.

- Select the template based on the language that you want to use:
  C#, **Visual Basic**, **C++** or **JavaScript**.

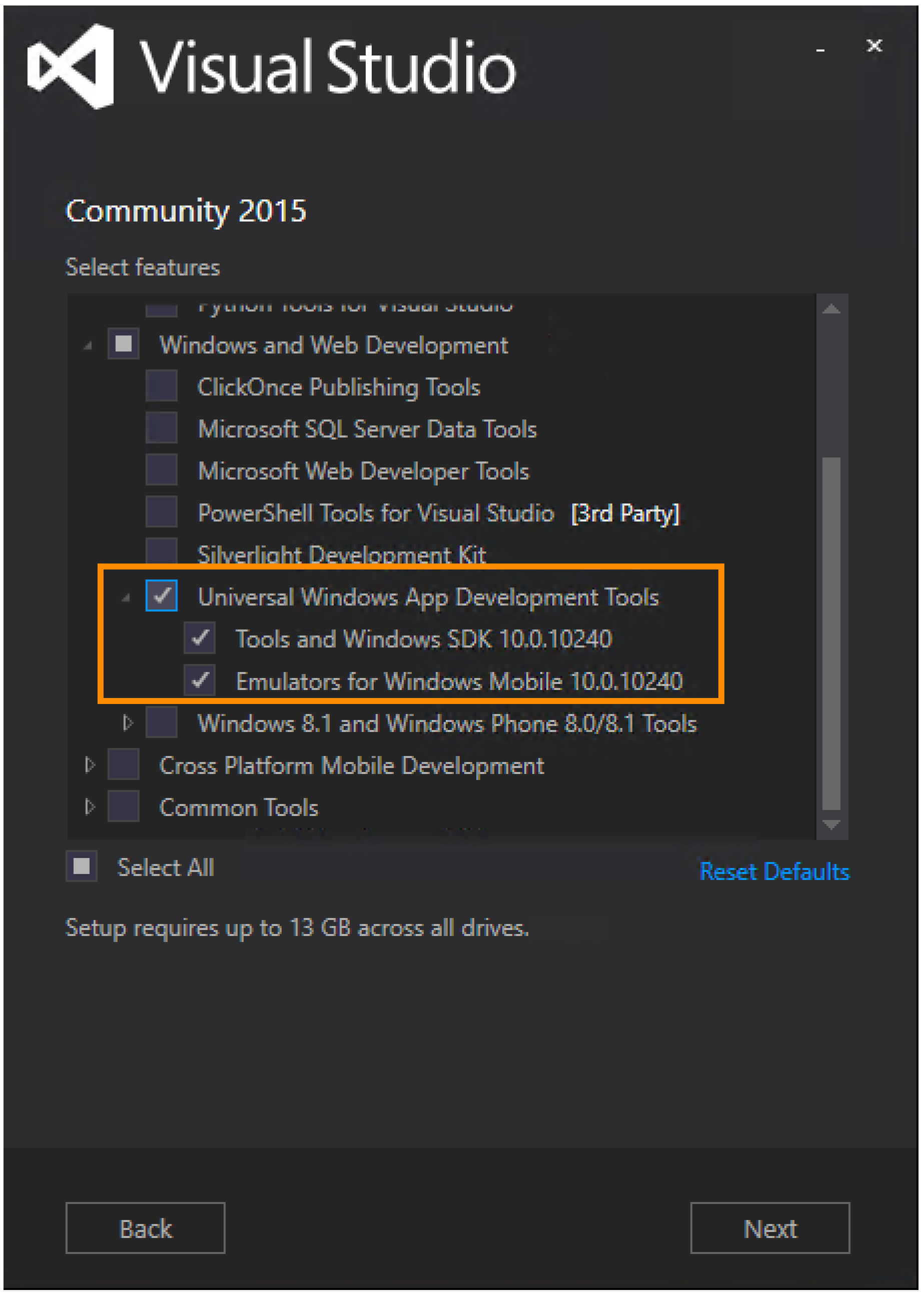
- Next create a Blank App (Universal Windows).

- Select the Target and Minimum version of Windows 10 suitable for your application.

  Click here if you are not sure which versions you should choose or simply leave the options at their default values and click 'OK' to get started!

Snapshots

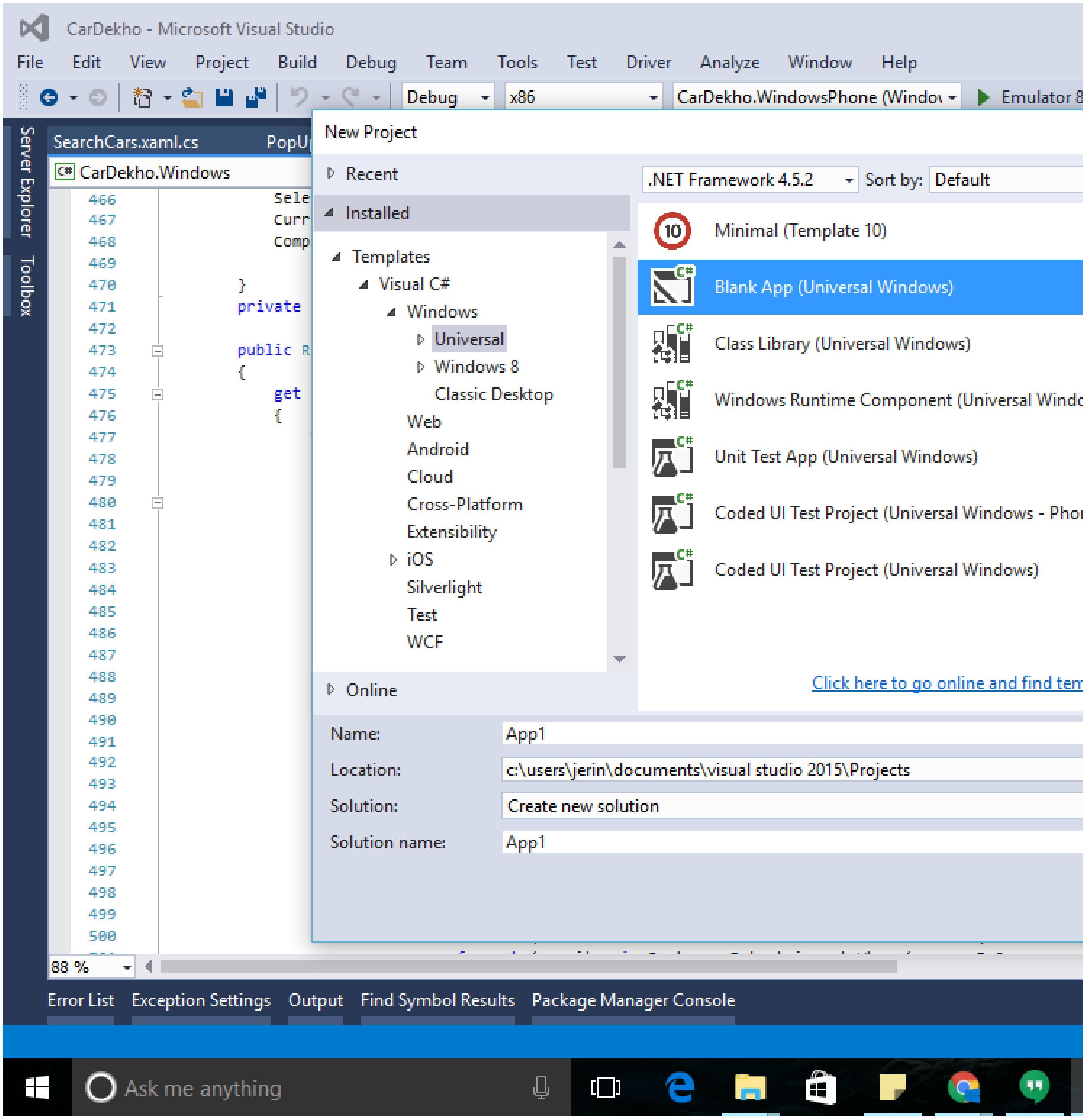
```
Installation
```



- [ ] Windows and Web Development
  - [ ] ClickOnce Publishing Tools
  - [ ] Microsoft SQL Server Data Tools
  - [ ] Microsoft Web Developer Tools
  - [ ] PowerShell Tools for Visual Studio [3rd Party]
  - [ ] Silverlight Development Kit
  - [✓] Universal Windows App Development Tools
    - [✓] Tools and Windows SDK 10.0.10240
    - [✓] Emulators for Windows Mobile 10.0.10240
  - [ ] Windows 8.1 and Windows Phone 8.0/8.1 Tools
- [ ] Cross Platform Mobile Development
- [ ] Common Tools

Setup requires up to 13 GB across all drives.

```
Creating a new project
```


CarDekho - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Driver Analyze Window Help
Debug
x86
CarDekho.WindowsPhone (Windov
Emulator 8
Server Explorer
Toolbox
SearchCars.xaml.cs
PopU
CarDekho.Windows

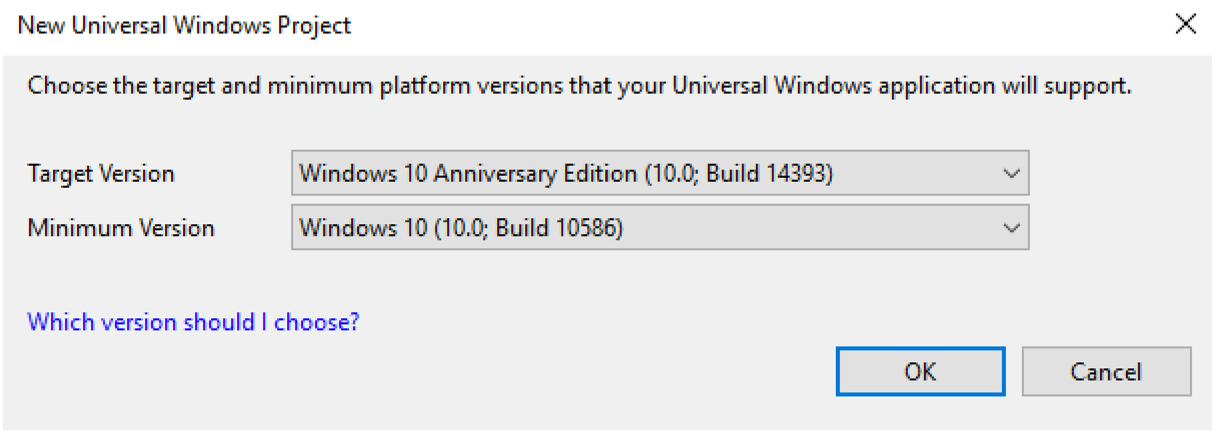
```
                Sele
                Curr
                Comp
            }
            private

            public R
            {
                get
                {
```

New Project
Recent
Installed
Templates
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
Android
Cloud
Cross-Platform
Extensibility
iOS
Silverlight
Test
WCF
Online
.NET Framework 4.5.2
Sort by: Default
Minimal (Template 10)
Blank App (Universal Windows)
Class Library (Universal Windows)
Windows Runtime Component (Universal Windo
Unit Test App (Universal Windows)
Coded UI Test Project (Universal Windows - Phon
Coded UI Test Project (Universal Windows)
Click here to go online and find tem

Name: App1
Location: c:\users\jerin\documents\visual studio 2015\Projects
Solution: Create new solution
Solution name: App1

88 %
Error List
Exception Settings
Output
Find Symbol Results
Package Manager Console
Ask me anything

```
Selecting Target and minimum version for your Application
```


New Universal Windows Project

Choose the target and minimum platform versions that your Universal Windows application will support.

Target Version: Windows 10 Anniversary Edition (10.0; Build 14393)

Minimum Version: Windows 10 (10.0; Build 10586)

Which version should I choose?

Creating your first UWP Application

This example demonstrates how to develop a simple UWP application.

On creation of a "Blank App (Universal Windows)" project there are many essential files that are created in your solution.

All files in your project can be seen in the **Solution Explorer**.

Some of the crucial files in your project are :

- **App.xaml and App.xaml.cs** - App.xaml is used to declare resources that are available across the application and App.xaml.cs is the backend code for it. App.xaml.cs is the default entry point of the application
- **MainPage.xaml** - This is the default startup UI page for your application (you can also change your application startup page in App.xaml.cs)
- **Package.appxmanifest** - This file contains important information of your application like Display name,entry point,visual assets,list of capabilities,packaging information etc.

Getting started

- **Adding a button to your page**

  To add any UI element or tool to your page simply drag and drop the element from the toolbox window on the left. Search for a "Button" tool in the toolbox and drop it in your app page.

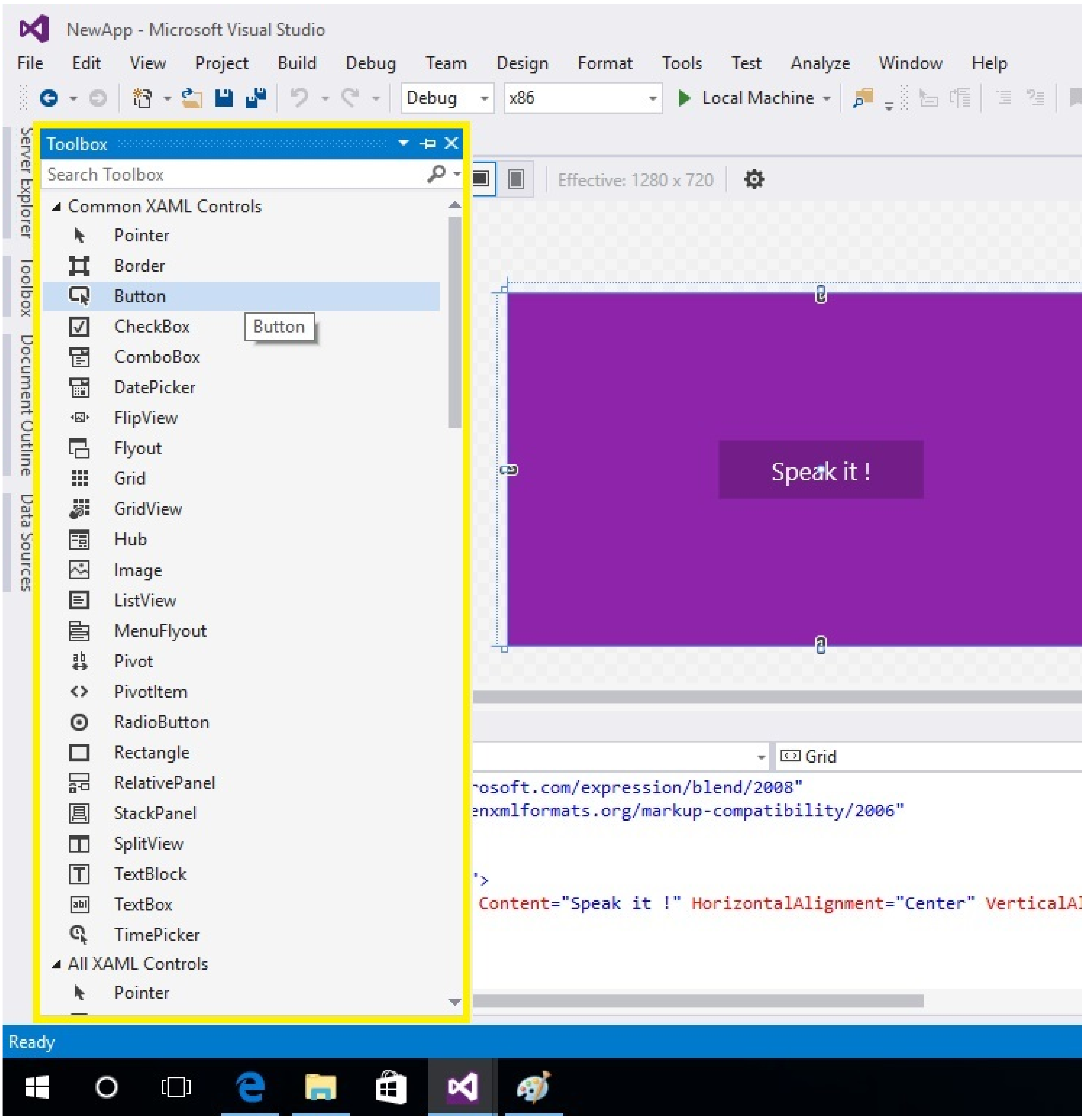
- **Customizing the UI**

  All properties for a particular tool is shown in the properties window on the Bottom Right side.

  Here we will change the text inside the button to "Speak it !". To do this first tap on the button to select it and then scroll through the properties window to find **Content** and change the text to your desired string ("Speak it !").

We will also change the background colour for the page. Each page has a parent element (usually a grid) which contains all the other elements . Thus we will change the colour of the parent grid. To do this tap on the grid and change the **Brush > Background** from the properties window to your desired colour.

The UI will look something like this after you have customized it .



```
rosoft.com/expression/blend/2008"
enxmlformats.org/markup-compatibility/2006"

'>
 Content="Speak it !" HorizontalAlignment="Center" VerticalAl
```

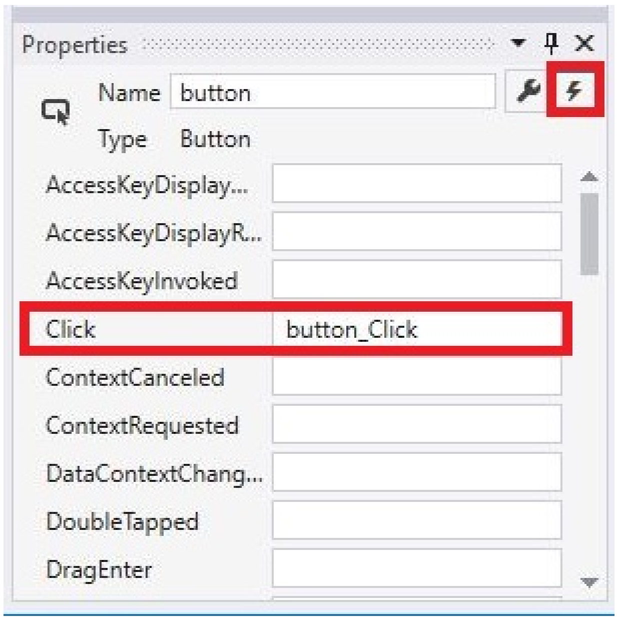
- **Code behind**

Now lets do something on click of our button!

Clicking on a button triggers an event and we need to handle the event to do something useful when the button is clicked.

Adding event handler

To add a click event handler to your button , select the button go to the properties window and select the ***lightning bolt icon*** . This window consists of all the events that are available for the element that we selected (the button in our case). Next, double click on the textbox beside *"Click"* event to auto-generate the handler for the button click event.



After this you will be redirected to a c# page (MainPage.xaml.cs). Add the following code to your event handler method:

```
 MediaElement mediaElement = new MediaElement();
        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
        Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream = await
synth.SynthesizeTextToStreamAsync("Hello, World!");
        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();
```

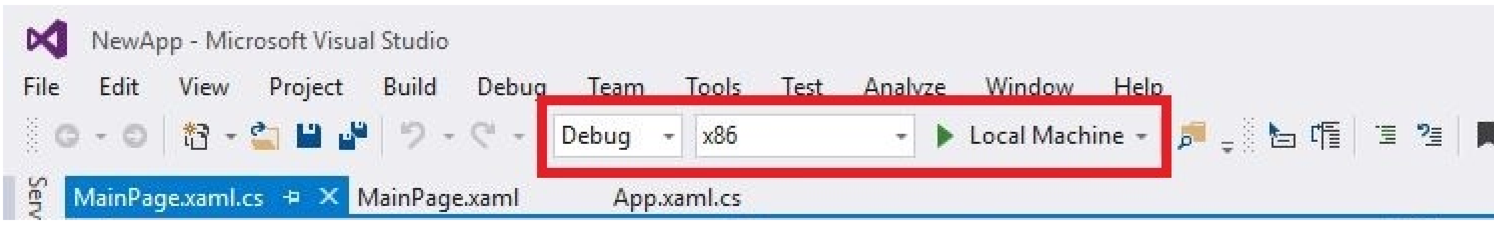
Next, add **async** keyword to your event handler.

After adding the code above your class should look something like this:

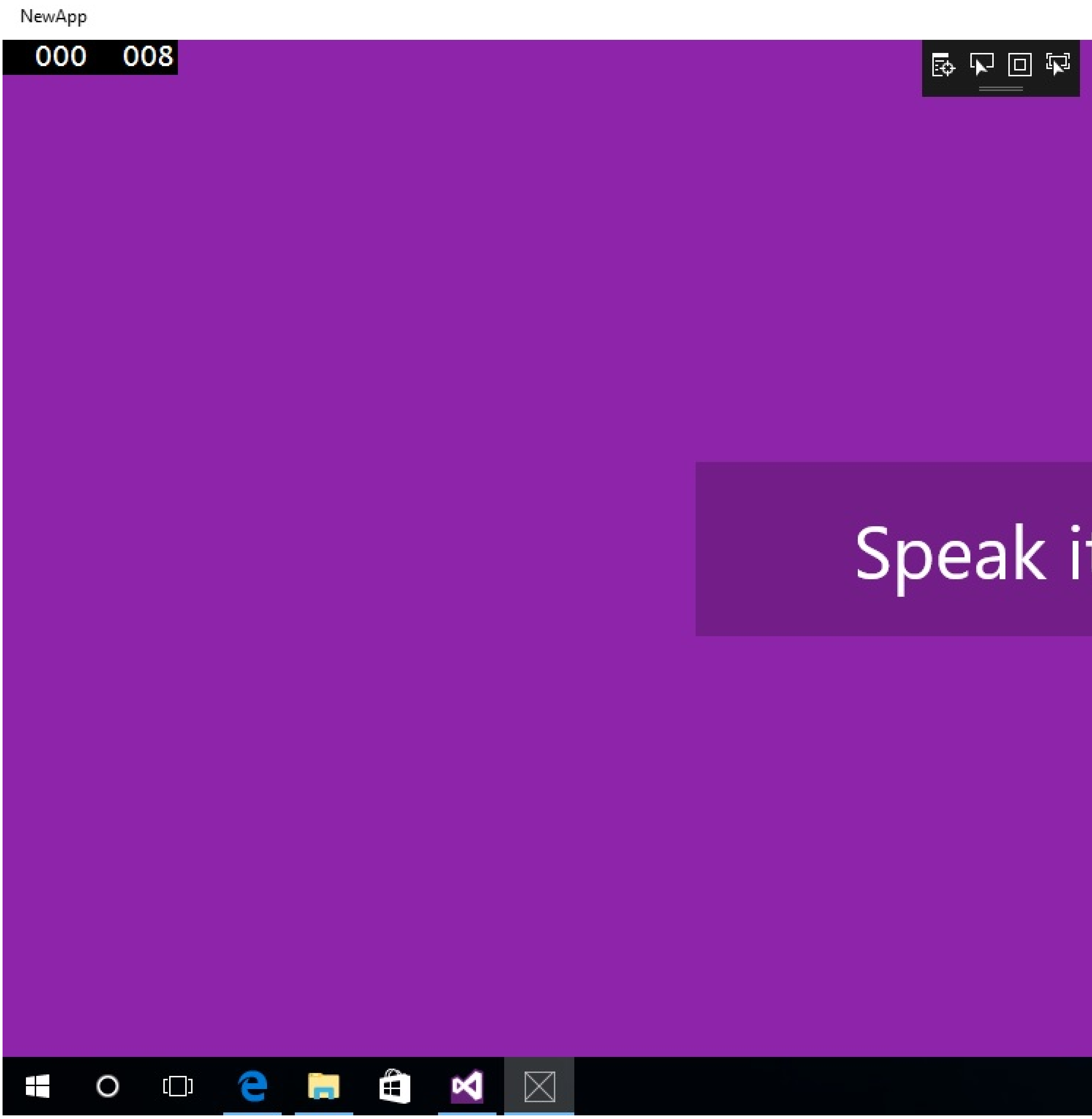
```
public sealed partial class MainPage : Page
{
    string speakIt = "Hello, World!";
    public MainPage()
    {
        this.InitializeComponent();
    }

    private async void button_Click(object sender, RoutedEventArgs e)
    {
        MediaElement mediaElement = new MediaElement();
        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
        Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream = await
synth.SynthesizeTextToStreamAsync(speakIt);
        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();

    }
}
```

- **Launch your app!**

  Your application is ready to be launched. You can launch your application by pressing F5 or Select your device on which you want to deploy and debug your application and click on start button.



After getting built, your application will be deployed on to your device. Depending on your device's resolution and screen size the application will automatically configure its layout. *(You can resize the window to see how seamlessly it works)*



- **Going further**

  Now that you have made your first application, let's go a step further !

  Add a textbox to your page and on click of the button, the app will speak out whatever is written in the textbox.

  Start by dragging and dropping a TextBox control from the Toolbox to your layout. Next, give a name to your TextBox from the properties menu. *(why do we need to specify a name ? so that we can easily use this control)*

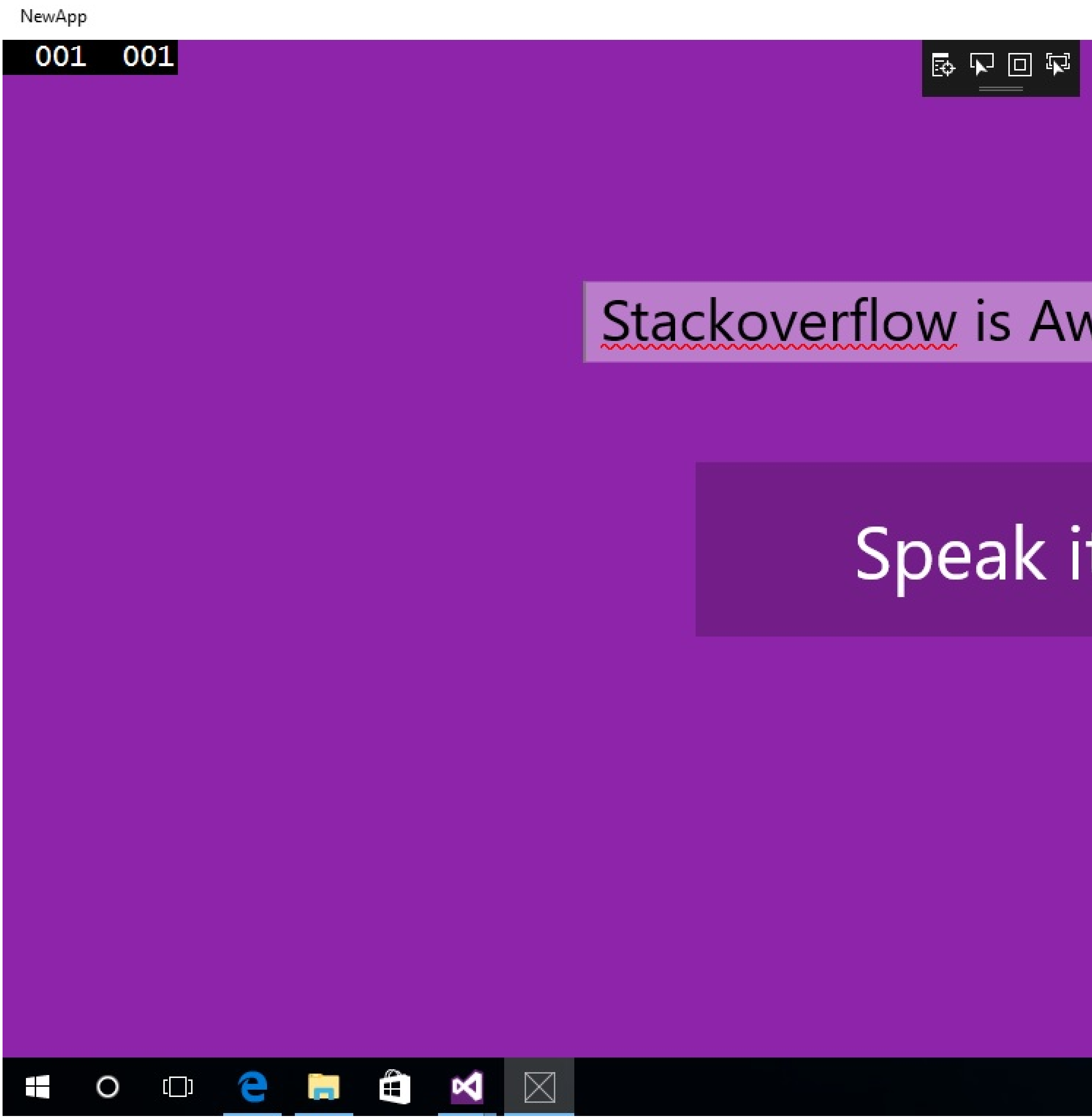
Visual Studio by default gives your control a name, but it's a good habit to name controls according to what they do or something relevant.

I am naming my textbox - "*speakText*".

```
    private async void button_Click(object sender, RoutedEventArgs e)
    {
        //checking if the text provided in the textbox is null or whitespace
        if (!string.IsNullOrWhiteSpace(speakText.Text))
            speakIt = speakText.Text;
        else
            speakIt = "Please enter a valid string!";

        MediaElement mediaElement = new MediaElement();
        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
        Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream = await
synth.SynthesizeTextToStreamAsync(speakIt);
        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();

    }
```

Now deploy your code!!

Your application is now able to speak out any *valid* string you provide to it !!



Congratulations ! You have successfully built your own UWP application !!

Read Getting started with uwp online: https://riptutorial.com/uwp/topic/1069/getting-started-with-uwp

Chapter 2: Adaptive UI

Examples

Use the AdaptiveTrigger to change the UI layout

The UWP applications can run in windowed mode and on several devices. They can be displayed on a wide range of screen sizes from low end phones to the huge surface hub screen. Using relative positioning will be enough for a lot of scenario but as the window size increases, it is always interesting to completely change the layout by moving the controls of page to different locations.

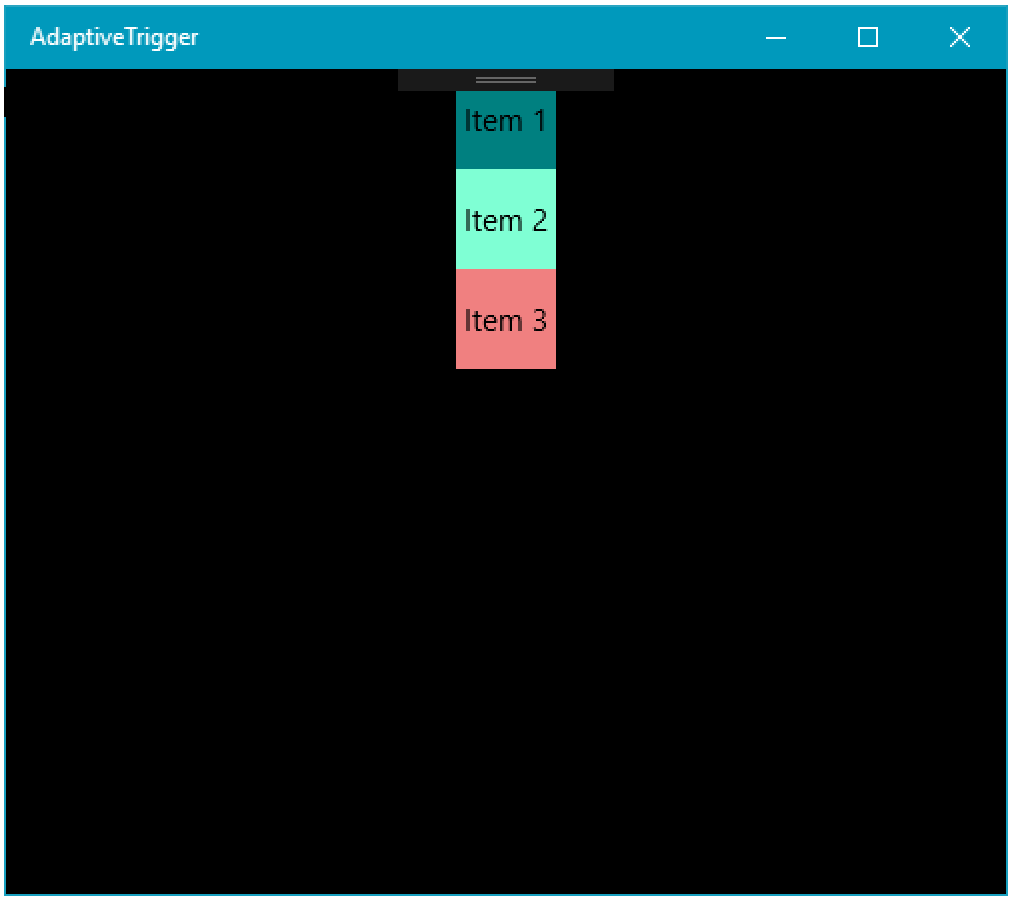
In this sample, we will use a vertical layout on narrow screens and an horizontal layout on wide screen. On huge wide screens, we will also change the items' sizes.

```
<Border x:Name="item2"
        Background="Aquamarine"
        Width="50"
        Height="50">
    <TextBlock Text="Item 2"
               VerticalAlignment="Center"
               HorizontalAlignment="Center" />
</Border>

<Border x:Name="item3"
        Background="LightCoral"
        Width="50"
        Height="50">
    <TextBlock Text="Item 3"
               VerticalAlignment="Center"
               HorizontalAlignment="Center" />
</Border>

<VisualStateManager.VisualStateGroups>
    <VisualStateGroup>

        <VisualState x:Name="ultrawide">
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="800" />
            </VisualState.StateTriggers>

            <VisualState.Setters>
                <Setter Target="mainPanel.Orientation" Value="Horizontal" />
                <Setter Target="item1.Width" Value="100" />
                <Setter Target="item1.Height" Value="100" />
                <Setter Target="item2.Width" Value="100" />
                <Setter Target="item2.Height" Value="100" />
                <Setter Target="item3.Width" Value="100" />
                <Setter Target="item3.Height" Value="100" />
            </VisualState.Setters>
        </VisualState>

        <VisualState x:Name="wide">
            <VisualState.StateTriggers>
```

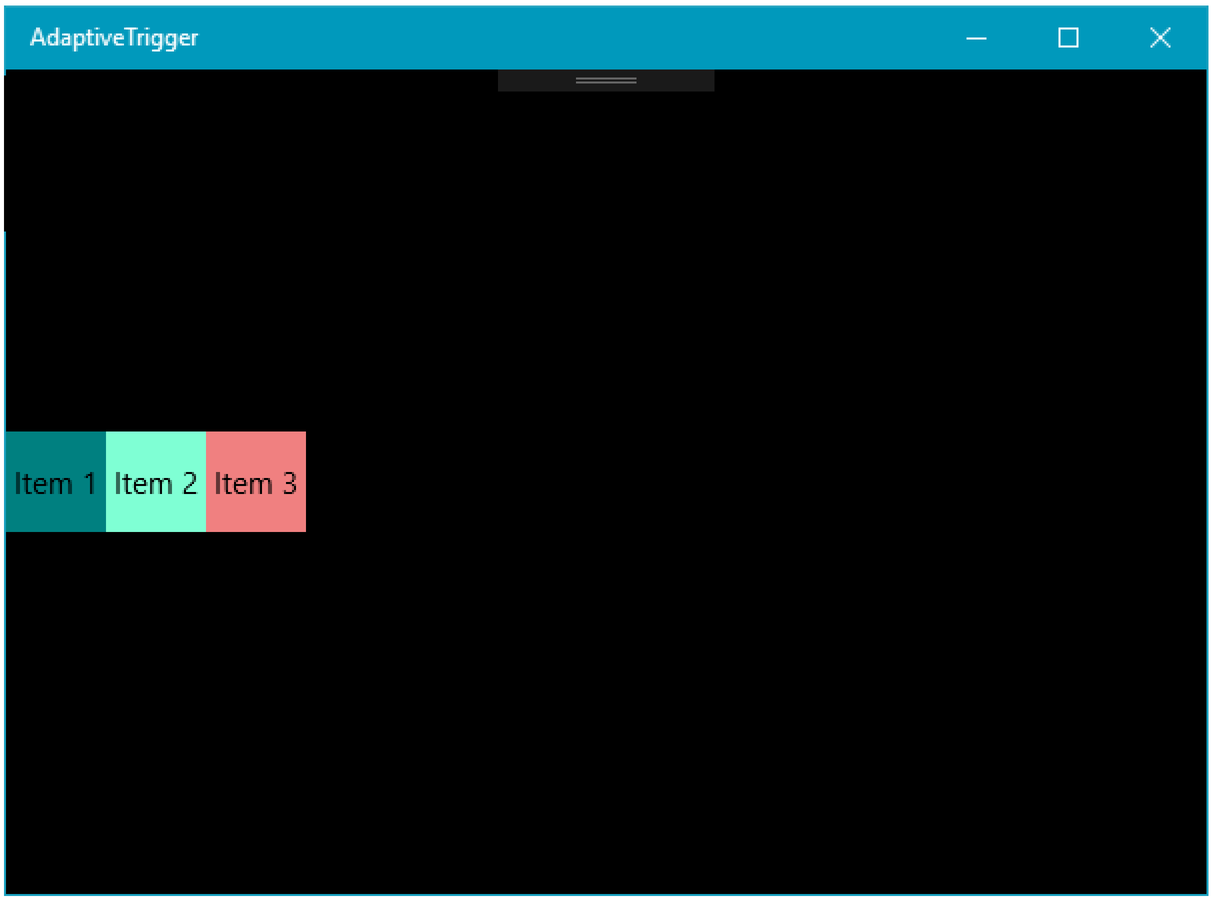
```
                <AdaptiveTrigger MinWindowWidth="600" />
            </VisualState.StateTriggers>

            <VisualState.Setters>
                <Setter Target="mainPanel.Orientation" Value="Horizontal" />
            </VisualState.Setters>
        </VisualState>

        <VisualState x:Name="narrow" />
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

When the window is resized, the system will compare the current window's width with the minimum width from the AdaptiveTrigger. If the current width is greater or equal than the minimum width, the trigger will be activated and the corresponding VisualState being displayed.

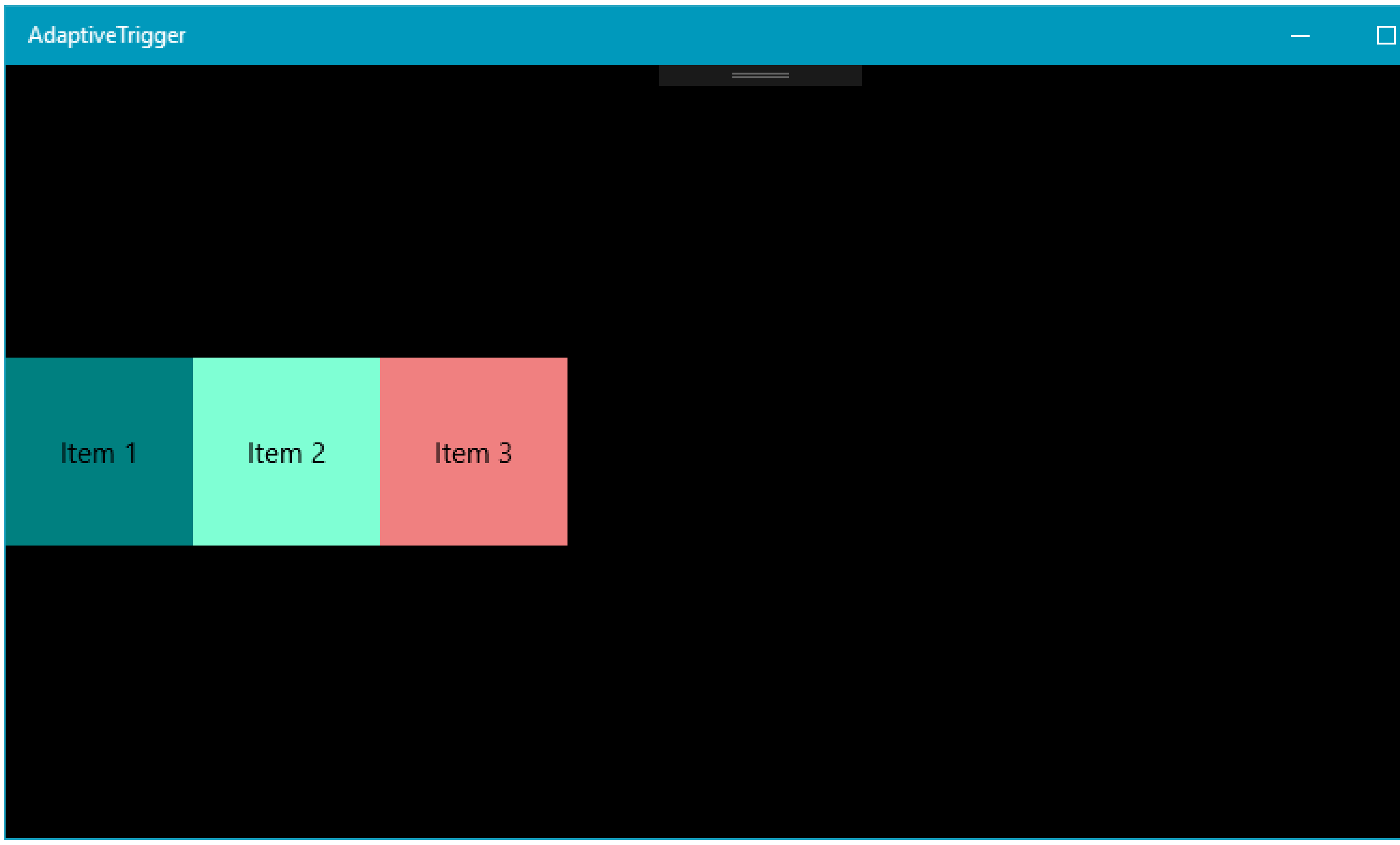
Here is the output for the different states

Narrow 

Wide



Ultrawide



Read Adaptive UI online: https://riptutorial.com/uwp/topic/6420/adaptive-ui

Chapter 3: Application Lifecycle

Introduction

Universal Windows 10 App lifecycle consists of three different states: 1) Running - application is currentyl in use 2) Not running - application is closed and removed from the memory 3) Suspended - application state is frozen but it is still in memory [![enter image description here][1]][1] [1]: https://i.stack.imgur.com/x7MCl.png As you cann see in the picture above there are different events connected with moving from one state to another. In examples section I show how to handle them.

Remarks

It is good to refer to two good articles on MSDN Blog:

1. https://msdn.microsoft.com/en-us/windows/uwp/launch-resume/app-lifecycle
2. https://blogs.windows.com/buildingapps/2016/04/28/the-lifecycle-of-a-uwp-app/#RqKAKkevsAPIvBUT.97

Examples

"Running" state handling

When moving to "Running" state there is special handler connected with this event: Open "App.xaml.cx" class and see "OnLaunched" method - this is activated when applicaiton is opened by user from "Terminaded" state:

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
    {
        Frame rootFrame = Window.Current.Content as Frame;

        // Do not repeat app initialization when the Window already has content,
        // just ensure that the window is active
        if (rootFrame == null)
        {
            // Create a Frame to act as the navigation context and navigate to the first page
            rootFrame = new Frame();

            rootFrame.NavigationFailed += OnNavigationFailed;

           //You can get information about previous state of the app:

            if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
            {
                //The app was previously suspended but was then shutdown
                //at some point because the system needed to reclaim memory.
            }
            if (e.PreviousExecutionState == ApplicationExecutionState.ClosedByUser)
            {
                //The user closed the app with the close gesture in tablet mode,
                //or with Alt+F4.When the user closes the app, it is first suspended
                //and then terminated.
            }
            if (e.PreviousExecutionState == ApplicationExecutionState.NotRunning)
            {
                //An app could be in this state because it hasn't been launched since the last
time
                //the user rebooted or logged in. It can also be in this state if it was
running
                //but then crashed, or because the user closed it earlier.
            }
            if (e.PreviousExecutionState == ApplicationExecutionState.Running)
            {
                //The app was already open when the user tried to launch it again
            }
            if (e.PreviousExecutionState == ApplicationExecutionState.Suspended)
            {
                //The user either minimized or switched away from your app
                //and didn't return to it within a few seconds.
            }

            // Place the frame in the current Window
            Window.Current.Content = rootFrame;
        }

        //When available system resources allow, the startup performance of Windows Store
        //apps on desktop device family devices is improved by proactively launching
        //the user’s most frequently used apps in the background. A prelaunched app
        //is put into the suspended state shortly after it is launched.Then, when the
        //user invokes the app, the app is resumed by bringing it from the suspended
        //state to the running state--which is faster than launching the app cold.
        //The user's experience is that the app simply launched very quickly.
        if (e.PrelaunchActivated == false)
        {
            if (rootFrame.Content == null)
            {
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            Window.Current.Activate();
        }
    }
```

"Suspending" state handling

When moving to "Suspened" state there is special handler connected with this event: Open "App.xaml.cx" class and see "App" constructor - there is event handler:

```
public App()
    {
        this.InitializeComponent();
        //Handle suspending operation with event handler:
        this.Suspending += OnSuspending;
    }
```

Now you can handle suspension event:

```
private Dictionary<string, object> _store = new Dictionary<string, object>();
private readonly string _saveFileName = "store.xml";
private async void OnSuspending(object sender, SuspendingEventArgs e)
    {
        var deferral = e.SuspendingOperation.GetDeferral();
        _store.Add("timestamp", DateTime.Now);
        await SaveStateAsync();
        //TODO: Save application state and stop any background activity
        //Here you can use  await SuspensionManager.SaveAsync();
        //To read more about saving state please refer to below MSDN Blog article:
        //https://blogs.windows.com/buildingapps/2016/04/28/the-lifecycle-of-a-uwp-
app/#RqKAKkevsAPIvBUT.97
        deferral.Complete();
    }

    private async Task SaveStateAsync()
    {
        var ms = new MemoryStream();
        var serializer = new DataContractSerializer(typeof(Dictionary<string, object>));
        serializer.WriteObject(ms, _store);

        var file = await ApplicationData.Current.LocalFolder.CreateFileAsync(_saveFileName,
CreationCollisionOption.ReplaceExisting);

        using (var fs = await file.OpenStreamForWriteAsync())
        {
            //because we have written to the stream, set the position back to start
            ms.Seek(0, SeekOrigin.Begin);
            await ms.CopyToAsync(fs);
            await fs.FlushAsync();
        }
    }
```

"Resuming" state handling

Your application can be opened by user from "Suspended" state. When doing it "OnResuming" event handler is used. In "App.xaml.cs" class:

```
public App()
    {
        this.InitializeComponent();
        this.Suspending += OnSuspending;
        //Handle resuming operation:
        this.Resuming += App_Resuming;
    }

private void App_Resuming(object sender, object e)
    {
        //Do some operation connected with app resuming for instance refresh data
    }
```

Read Application Lifecycle online: https://riptutorial.com/uwp/topic/8135/application-lifecycle

Chapter 4: Binding vs x:Bind

Syntax

- <object property="{x:Bind}" .../>
- <object property="{x:Bind propertyPath}" .../>
- <object property="{x:Bind Path=propertyPath}" .../>
- <object property="{x:Bind [bindingProperties]}" .../>
- <object property="{x:Bind propertyPath, [bindingProperties]}" .../>
- <object property="{x:Bind Path=propertyPath, [bindingProperties]}" .../>

Remarks

The `{x:Bind}` markup extension—new for Windows 10—is an alternative to `{Binding}`.

`{x:Bind}`lacks some of the features of `{Binding}`, but it runs in less time and less memory than `{Binding}` and supports better debugging.

At XAML load time, `{x:Bind}` is converted into what you can think of as a binding object, and this object gets a value from a property on a data source. The binding object can optionally be configured to observe changes in the value of the data source property and refresh itself based on those changes. It can also optionally be configured to push changes in its own value back to the source property. The binding objects created by `{x:Bind}` and `{Binding}` are largely functionally equivalent. But `{x:Bind}` executes special-purpose code, which it generates at compile-time, and `{Binding}` uses general-purpose runtime object inspection. Consequently, `{x:Bind}` bindings (often referred-to as compiled bindings) have great performance, provide compile-time validation of your binding expressions, and support debugging by enabling you to set breakpoints in the code files that are generated as the partial class for your page. These files can be found in your obj folder, with names like (for C#) .g.cs.

For more information please see the MSDN documentation on x:Bind

Examples

Evaluating {x:Bind} from functions

This ability was added to the Bind markup extension after v1607 (Redstone 1).
You can specify a function path, as well as arg paths and constant args. Let's assume we have a function defined in our backcode:

```
public string Translate(string text, string from, string to)
```

Now we can use bind to evaluate the function into the element we want:

```
<TextBlock Name="SomeText" Text="How are you?" />
<TextBlock Name="{x:Bind Translate(SomeText.Text, 'en', 'es')}" />
```

Function and arg paths can contain dots and casts as well.

Read Binding vs x:Bind online: https://riptutorial.com/uwp/topic/4951/binding-vs-x-bind

Chapter 5: Binding vs x:Bind

Remarks

Refer the official Data binding documentation from Microsoft.

Examples

Binding modes and defaults

There are three modes of XAML bindings exists for either `Binding` and `x:Bind`:

- **OneTime**: Update happens only once, on initialization of the view during `InitializeComponent()` call. *(ViewModel[sends data when initializing] -> View)*
- **OneWay**: View is updated when ViewModel changes. But not in the reverse direction. *(ViewModel -> View)*
- **TwoWay**: View is updated when ViewModel changes and vice versa. *(ViewModel <-> View)*

Default mode of `Binding` is `OneWay` and that of `x:Bind` is `OneTime`.

Select the modes like this:

```
<TextBlock Text="{Binding SomeText, Mode=TwoWay}" /> <!-- Binding -->
<TextBlock Text="{x:Bind SomeText, Mode=OneWay}" /> <!-- x:Bind -->
```

When to use x:Bind

- When calling methods directly from the view.
- If performance matters really bad (scientific spaceship stuff)
- When you want to get compile time errors

When to use Binding

- Use it if you want to be flexible about the source type of your data. It won't bind to an actual property but to its name.
- If you want to bind to the DataContext

Read Binding vs x:Bind online: https://riptutorial.com/uwp/topic/6412/binding-vs-x-bind

Chapter 6: Convert image size and crop image file in Windows Universal app

Examples

Crop and resize image using bitmap tool

```
 public class BitmapTools
    {
        /// <summary>
        /// Gets the cropped bitmap asynchronously.
        /// </summary>
        /// <param name="originalImage">The original image.</param>
        /// <param name="startPoint">The start point.</param>
        /// <param name="cropSize">Size of the corp.</param>
        /// <param name="scale">The scale.</param>
        /// <returns>The cropped image.</returns>
        public static async Task<WriteableBitmap> GetCroppedBitmapAsync(IRandomAccessStream
originalImage,
            Point startPoint, Size cropSize, double scale)
        {
            if (double.IsNaN(scale) || double.IsInfinity(scale))
            {
                scale = 1;
            }

            // Convert start point and size to integer.
            var startPointX = (uint)Math.Floor(startPoint.X * scale);
            var startPointY = (uint)Math.Floor(startPoint.Y * scale);
            var height = (uint)Math.Floor(cropSize.Height * scale);
            var width = (uint)Math.Floor(cropSize.Width * scale);

            // Create a decoder from the stream. With the decoder, we can get
            // the properties of the image.
            var decoder = await BitmapDecoder.CreateAsync(originalImage);

            // The scaledSize of original image.
            var scaledWidth = (uint)Math.Floor(decoder.PixelWidth * scale);
            var scaledHeight = (uint)Math.Floor(decoder.PixelHeight * scale);

            // Refine the start point and the size.
            if (startPointX + width > scaledWidth)
            {
                startPointX = scaledWidth - width;
            }

            if (startPointY + height > scaledHeight)
            {
                startPointY = scaledHeight - height;
            }

            // Get the cropped pixels.
            var pixels = await GetPixelData(decoder, startPointX, startPointY, width, height,
                scaledWidth, scaledHeight);
            // Stream the bytes into a WriteableBitmap
            var cropBmp = new WriteableBitmap((int)width, (int)height);
            var pixStream = cropBmp.PixelBuffer.AsStream();
            pixStream.Write(pixels, 0, (int)(width * height * 4));

            return cropBmp;
        }

        /// <summary>
        /// Gets the pixel data.
        /// </summary>
        /// <remarks>
        /// If you want to get the pixel data of a scaled image, set the scaledWidth and
scaledHeight
        /// of the scaled image.
        /// </remarks>
        /// <param name="decoder">The bitmap decoder.</param>
        /// <param name="startPointX">The X coordinate of the start point.</param>
        /// <param name="startPointY">The Y coordinate of the start point.</param>
        /// <param name="width">The width of the source rect.</param>
        /// <param name="height">The height of the source rect.</param>
        /// <param name="scaledWidth">The desired width.</param>
        /// <param name="scaledHeight">The desired height.</param>
        /// <returns>The image data.</returns>
        private static async Task<byte[]> GetPixelData(BitmapDecoder decoder, uint
startPointX, uint startPointY,
            uint width, uint height, uint scaledWidth, uint scaledHeight)
        {
            var transform = new BitmapTransform();
            var bounds = new BitmapBounds();
            bounds.X = startPointX;
            bounds.Y = startPointY;
            bounds.Height = height;
            bounds.Width = width;
            transform.Bounds = bounds;

            transform.ScaledWidth = scaledWidth;
            transform.ScaledHeight = scaledHeight;

            // Get the cropped pixels within the bounds of transform.
            var pix = await decoder.GetPixelDataAsync(
                BitmapPixelFormat.Bgra8,
                BitmapAlphaMode.Straight,
                transform,
                ExifOrientationMode.IgnoreExifOrientation,
                ColorManagementMode.ColorManageToSRgb);
            var pixels = pix.DetachPixelData();
            return pixels;
        }

        /// <summary>
        /// Resizes the specified stream.
        /// </summary>
        /// <param name="sourceStream">The source stream to resize.</param>
        /// <param name="newWidth">The width of the resized image.</param>
        /// <param name="newHeight">The height of the resized image.</param>
        /// <returns>The resized image stream.</returns>
        public static async Task<InMemoryRandomAccessStream> Resize(IRandomAccessStream
sourceStream, uint requestedMinSide)
        {
            var decoder = await BitmapDecoder.CreateAsync(sourceStream);
            uint originalPixelWidth = decoder.OrientedPixelWidth;
            uint originalPixelHeight = decoder.OrientedPixelHeight;

            double widthRatio = (double)requestedMinSide / originalPixelWidth;
            double heightRatio = (double)requestedMinSide / originalPixelHeight;
            uint aspectHeight = (uint)requestedMinSide;
            uint aspectWidth = (uint)requestedMinSide;
            var scaledSize = (uint)requestedMinSide;
            if (originalPixelWidth < originalPixelHeight)
            {
                aspectWidth = (uint)(heightRatio * originalPixelWidth);
            }
            else
            {
                aspectHeight = (uint)(widthRatio * originalPixelHeight);
            }

            var destinationStream = new InMemoryRandomAccessStream();


            var transform = new BitmapTransform { ScaledWidth = aspectWidth, ScaledHeight =
aspectHeight };

            var pixelData = await decoder.GetPixelDataAsync(
                BitmapPixelFormat.Bgra8,
                BitmapAlphaMode.Straight,
                transform,
                ExifOrientationMode.RespectExifOrientation,
                ColorManagementMode.DoNotColorManage);

            var encoder =
                await BitmapEncoder.CreateAsync(BitmapEncoder.JpegEncoderId,
destinationStream);

            if(decoder.OrientedPixelHeight!=decoder.PixelHeight &&
decoder.OrientedPixelWidth!=decoder.PixelWidth)
            encoder.BitmapTransform.Rotation = BitmapRotation.Clockwise270Degrees;

            encoder.SetPixelData(BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied,
aspectWidth, aspectHeight, 96, 96,
                pixelData.DetachPixelData());
            await encoder.FlushAsync();


            return destinationStream;
        }

        /// <summary>
        /// Rotates the given stream.
        /// </summary>
        /// <param name="randomAccessStream">The random access stream.</param>
        /// <param name="rotation">The rotation.</param>
        /// <returns>The stream.</returns>
        public static async Task<InMemoryRandomAccessStream> Rotate(IRandomAccessStream
randomAccessStream,
            BitmapRotation rotation)
        {
            var decoder = await BitmapDecoder.CreateAsync(randomAccessStream);

            var rotatedStream = new InMemoryRandomAccessStream();
            var encoder = await BitmapEncoder.CreateForTranscodingAsync(rotatedStream,
decoder);

            encoder.BitmapTransform.Rotation = rotation;
            encoder.BitmapTransform.InterpolationMode = BitmapInterpolationMode.Fant;

            await encoder.FlushAsync();

            return rotatedStream;
        }



        /// <summary>
        /// Resizes and crops source file image so that resized image width/height are not
larger than <param name="requestedMinSide"></param>
        /// </summary>
        /// <param name="sourceFile">Source StorageFile</param>
        /// <param name="requestedMinSide">Width/Height of the output image</param>
        /// <param name="resizedImageFile">Target StorageFile</param>
        /// <returns></returns>
        public static async Task<IStorageFile> CreateThumbnaiImage(StorageFile sourceFile, int
requestedMinSide, StorageFile resizedImageFile)
        {
            var imageStream = await sourceFile.OpenReadAsync();
            var decoder = await BitmapDecoder.CreateAsync(imageStream);
            var originalPixelWidth = decoder.PixelWidth;
            var originalPixelHeight = decoder.PixelHeight;
            using (imageStream)
            {
                //do resize only if needed
                if (originalPixelHeight > requestedMinSide && originalPixelWidth >
requestedMinSide)
                {
                    using (var resizedStream = await
resizedImageFile.OpenAsync(FileAccessMode.ReadWrite))
                    {
                        //create encoder based on decoder of the source file
                        var encoder = await
BitmapEncoder.CreateForTranscodingAsync(resizedStream, decoder);
                        double widthRatio = (double)requestedMinSide / originalPixelWidth;
                        double heightRatio = (double)requestedMinSide / originalPixelHeight;
                        uint aspectHeight = (uint)requestedMinSide;
                        uint aspectWidth = (uint)requestedMinSide;
                        uint cropX = 0, cropY = 0;
                        var scaledSize = (uint)requestedMinSide;
                        if (originalPixelWidth > originalPixelHeight)
                        {
                            aspectWidth = (uint)(heightRatio * originalPixelWidth);
                            cropX = (aspectWidth - aspectHeight) / 2;
                        }
                        else
                        {
                            aspectHeight = (uint)(widthRatio * originalPixelHeight);
                            cropY = (aspectHeight - aspectWidth) / 2;
                        }
                        //you can adjust interpolation and other options here, so far linear
is fine for thumbnails
                        encoder.BitmapTransform.InterpolationMode =
BitmapInterpolationMode.Linear;
                        encoder.BitmapTransform.ScaledHeight = aspectHeight;
                        encoder.BitmapTransform.ScaledWidth = aspectWidth;
                        encoder.BitmapTransform.Bounds = new BitmapBounds()
                        {
                            Width = scaledSize,
                            Height = scaledSize,
                            X = cropX,
                            Y = cropY,
                        };
                        await encoder.FlushAsync();
                    }
                }
                else
                {
                    //otherwise just use source file as thumbnail
                    await sourceFile.CopyAndReplaceAsync(resizedImageFile);
                }
            }
            return resizedImageFile;
        }

    }
```

Read Convert image size and crop image file in Windows Universal app online: https://riptutorial.com/uwp/topic/9529/convert-image-size-and-crop-image-file-in-windows-universal-app

Chapter 7: Device Families

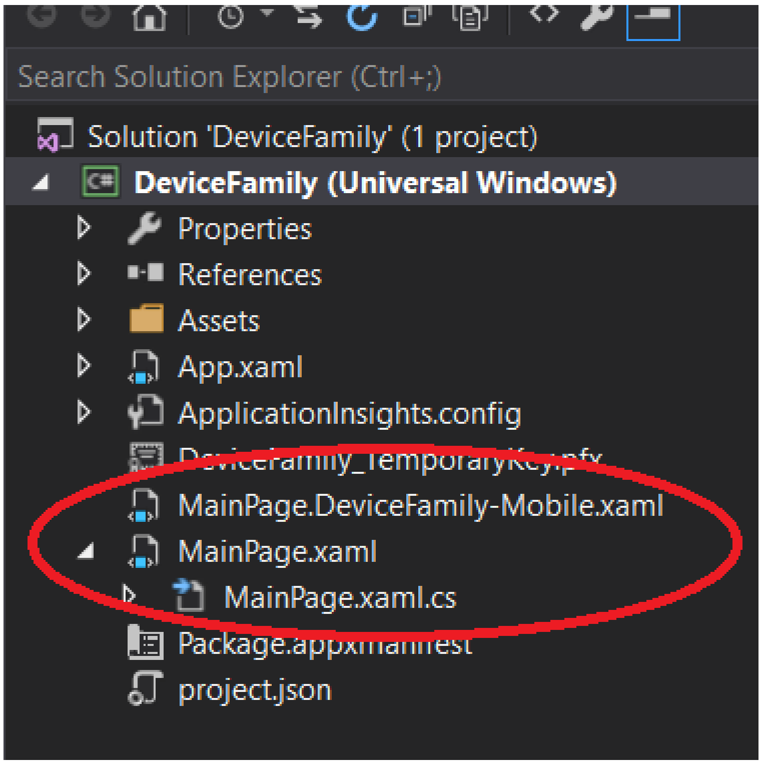
Examples

DeviceFamily specific code

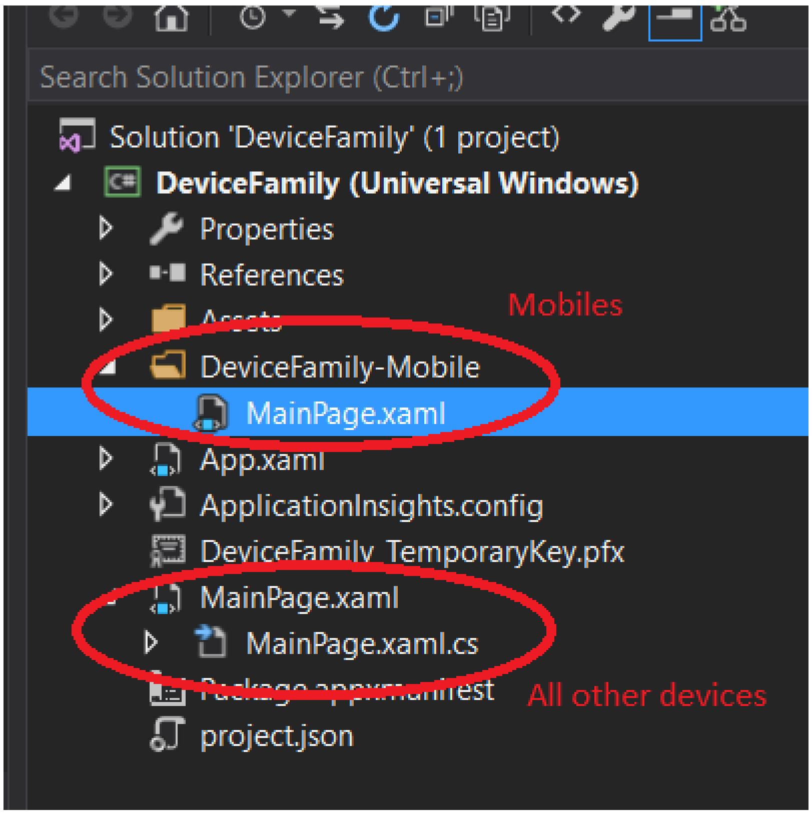
In general, UWP is used for making a single application that runs on Windows 10 across many different devices. However, it is also possible to make code tailored to specific devices. You can achieve this in several different ways.

Different XAML Layout

If you want to use a specific layout on for a certain "device family", you can do this by creating a new XAML Page item with the same name as the default XAML file, with a suffix to indicate the device family you are targeting. Then you'll have *MainPage.xaml* for all devices and *MainPage.DeviceFamily-[specific family].xaml* just for one specific family, which will overwrite the default layout, see below:


Search Solution Explorer (Ctrl+;)
Solution 'DeviceFamily' (1 project)
DeviceFamily (Universal Windows)
Properties
References
Assets
App.xaml
ApplicationInsights.config
DeviceFamily_TemporaryKey.pfx
MainPage.DeviceFamily-Mobile.xaml
MainPage.xaml
MainPage.xaml.cs
Package.appxmanifest
project.json

If you want to do this for lots of files, you can make a folder with name *DeviceFamily-[specific family]* and put all XAML pages into it, but now with exactly with the same name as the default XAML file (see below). In both examples, all pages would share the same code-behind file, so the functionality is identical, but the layout is tailored to specific screen sizes.


Search Solution Explorer (Ctrl+;)
Solution 'DeviceFamily' (1 project)
DeviceFamily (Universal Windows)
Properties
References
Assets
DeviceFamily-Mobile
MainPage.xaml
App.xaml
ApplicationInsights.config
DeviceFamily_TemporaryKey.pfx
MainPage.xaml
MainPage.xaml.cs
Package.appxmanifest
project.json
Mobiles
All other devices

Code for specific family

If you want to run part of your code-behind or your ViewModel on a specific device family only, you can use the `DeviceFamily` property from the `AnalyticsVersionInfo` class.

```
AnalyticsVersionInfo avi = AnalyticsInfo.VersionInfo;
var deviceFamily = avi.DeviceFamily;

if(deviceFamily == "Windows.Mobile")
{
   Console.WriteLine("You're on mobile device right now.");
}
else if(deviceFamily == "Windows.Desktop")
{
   Console.WriteLine("You're on desktop");
}
else if(deviceFamily == "Windows.IoT")
{
   Console.WriteLine("You're on IoT");
}
//....
```

Get current device family

Here a simple portable way to get the current device family:

```
/// <summary>
/// All the device families
/// </summary>
public enum DeviceFamily
{
    Desktop,
    Mobile,
    Iot,
    Xbox,
}

/// <summary>
/// The helper to get the current device family
/// </summary>
public static class DeviceFamilyHelper
{
    /// <summary>
    /// Return the family of the current device
    /// </summary>
    /// <returns>the family of the current device</returns>
    public static DeviceFamily GetDeviceFamily()
    {

switch(ResourceContext.GetForCurrentView().QualifierValues["DeviceFamily"].ToLowerInvariant())
        {
            case "mobile":  return DeviceFamily.Mobile;
            case "xbox":    return DeviceFamily.Xbox;
            case "iot":     return DeviceFamily.Iot;
            default:        return DeviceFamily.Desktop;
        }
    }
}
```

Detect if an API contract is supported

Depending on the device/release version of the system, some API may not be available. You can check which contract is supported by using ApiInformation.IsApiContractPresent()

For example, this will return true on phone devices and false on the others

```
ApiInformation.IsApiContractPresent(typeof(CallsPhoneContract).FullName, 1)
```

The contract where an API belong is available at the bottom the API page on the MSDN or the global list is available from the API contract page.

Read Device Families online: https://riptutorial.com/uwp/topic/3994/device-families

Chapter 8: File name qualifiers

Remarks

Qualifiers are used in this common format:

Files: `filename.qualifier-value.ext`
~ *multiple qualifiers:* `filename.qualifier1-value1_qualifier2-value2_....ext`

Qualified folders: `qualifier-value`
~ *multiple qualifiers:* `qualifier1-value1_qualifier2-value2_...`

Qualifiers are listed bellow, they are used in the format described above

Qualifier	Usage	Values
Lang / Language	Specifies a language, region or both.	`xx-xx`, or `xx` values in BCP-47
Scale	Qualifies the device scale factor.	*Commonly* 100 / 125 / 150 / 200 / 400
DeviceFamily	Specifies the device type.	Mobile / Team / Desktop / IoT
Contrast	Specifies the contrast theme type.	Standard / High / Black / White
HomeRegion	Specifies user's home region.	Any ISO3166-1 alpha2 or numeric code
TargetSize	Gives the smallest image larger than need.	Any positive integer.
LayoutDir	Specifies a layout direction.	RTL / LTR / TTBRTL / TTBLTR
Config	Qualifies for `MS_CONFIGURATION_ATTRIBUTE_VALUE`.	The value of environment config.
DXFL*	Specifies a DirectX feature level.	DX9 / DX10 / DX11

* Also used as **DXFeatureLevel**.

Some notes to keep in mind:

- `HomeRegion` won't accept groupings or unions.
- `TargetSize` and `Scale` cannot be used together.

Examples

Using different views for device types

You can qualify a whole folder folder for a specific device type, its files will override the ones outside it on that device:

```
/ DeviceFamily-Mobile
    PageOfEden.xaml
    MainPage.xaml
MainPage.xaml
MainPage.xaml.cs
PageOfEden.xaml
PageOfEden.xaml.cs
```

Files inside the qualifying folder won't need qualifiers.

Default asset scaling qualifiers

If you browse your app's *Assets* folder you will notice that all resources are qualified by their scales (As you are required to put seperate files for each scaling in the package manifest).

```
SplashScreen.scale-100.png
SplashScreen.scale-125.png
SplashScreen.scale-150.png
SplashScreen.scale-200.png
```

Using the TargetSize qualifier

Let's assume we have an *Image* element using a square image named `Picture.png`.
We can use different files for each dimension set for the element.

```
Picture.TargetSize-16.png
Picture.TargetSize-32.png
Picture.TargetSize-128.png
```

Now if we set the `Height` or `Width` of our *Image* to 16px, it will use `Picture.TargetSize-16.png` as a source. Now if we set the dimensions to 20px, there is no image matching the exact dimensions, so it will use `Picture.TargetSize-32.png`, as it's the nearest image larger than our needs. Dimensions higher than 128 will use `Picture.TargetSize-128.png`.

Read File name qualifiers online: https://riptutorial.com/uwp/topic/6733/file-name-qualifiers

Chapter 9: How to get current DateTime in C++ UWP

Introduction

The documentation for the `DateTime::UniversalTime` states:

"A 64-bit signed integer that represents a point in time as the number of 100-nanosecond intervals prior to or after midnight on January 1, 1601 (according to the Gregorian Calendar)."

This is the same as the Win32 `FILETIME`struct which you need to convert to a 100-nanosecond long long value and set it in the `DateTime::UniversalTime` field.

Examples

GetCurrentDateTime()

```
#include <windows.h>

static Windows::Foundation::DateTime GetCurrentDateTime() {
    // Get the current system time
    SYSTEMTIME st;
    GetSystemTime(&st);

    // Convert it to something DateTime will understand
    FILETIME ft;
    SystemTimeToFileTime(&st, &ft);

    // Conversion to DateTime's long long is done vie ULARGE_INTEGER
    ULARGE_INTEGER ui;
    ui.LowPart = ft.dwLowDateTime;
    ui.HighPart = ft.dwHighDateTime;

    DateTime currentDateTime;
    currentDateTime.UniversalTime = ui.QuadPart;
    return currentDateTime;
}
```

Read How to get current DateTime in C++ UWP online:
https://riptutorial.com/uwp/topic/10131/how-to-get-current-datetime-in-cplusplus-uwp

Chapter 10: Images

Parameters

Parameter	Description
DecodePixelWidth	Will load the BitmapImage with the specified width. Helps with memory usage and speed when loading large images that are meant to be displayed smaller on the screen. This is more efficient than loading full image and rely on the `Image` control to do the resize.
DecodePixelHeight	Same as `DecodePixelHeight`. If only one parameter is specified the system will maintain the Aspect Ratio of the image while loading at the required size.

Examples

Using BitmapImage with Image control

```
<Image x:Name="MyImage" />
```

```
// Show image from web
MyImage.Source = new BitmapImage(new Uri("http://your-image-url.com"))

// Show image from solution
MyImage.Source = new Uri("ms-appx:///your-image-in-solution", UriKind.Absolute)

// Show image from file
IRandomAccessStreamReference file = GetFile();
IRandomAccessStream fileStream = await file.OpenAsync();
var image = new BitmapImage();
await image.SetSourceAsync(fileStream);
MyImage.Source = image;
fileStream.Dispose(); // Don't forget to close the stream
```

Rendering controls to image with RenderTargetBitmap

```
<TextBlock x:Name="MyControl"
           Text="Hello, world!" />
```

```
var rtb = new RenderTargetBitmap();
await rtb.RenderAsync(MyControl); // Render control to RenderTargetBitmap

// Get pixels from RTB
IBuffer pixelBuffer = await rtb.GetPixelsAsync();
byte[] pixels = pixelBuffer.ToArray();

// Support custom DPI
```

```
DisplayInformation displayInformation = DisplayInformation.GetForCurrentView();

var stream = new InMemoryRandomAccessStream();
BitmapEncoder encoder = await BitmapEncoder.CreateAsync(BitmapEncoder.PngEncoderId, stream);
encoder.SetPixelData(BitmapPixelFormat.Bgra8, // RGB with alpha
                     BitmapAlphaMode.Premultiplied,
                     (uint)rtb.PixelWidth,
                     (uint)rtb.PixelHeight,
                     displayInformation.RawDpiX,
                     displayInformation.RawDpiY,
                     pixels);

await encoder.FlushAsync(); // Write data to the stream
stream.Seek(0); // Set cursor to the beginning

// Use stream (e.g. save to file)
```

Convert Bitmap (e.g. from Clipboard content) to PNG

```
IRandomAccessStreamReference bitmap = GetBitmap();
IRandomAccessStreamWithContentType stream = await bitmap.OpenReadAsync();
BitmapDecoder decoder = await BitmapDecoder.CreateAsync(stream);
var pixels = await decoder.GetPixelDataAsync();
var outStream = new InMemoryRandomAccessStream();

// Create encoder for PNG
var encoder = await BitmapEncoder.CreateAsync(BitmapEncoder.PngEncoderId, outStream);

// Get pixel data from decoder and set them for encoder
encoder.SetPixelData(decoder.BitmapPixelFormat,
                     BitmapAlphaMode.Ignore, // Alpha is not used
                     decoder.OrientedPixelWidth,
                     decoder.OrientedPixelHeight,
                     decoder.DpiX, decoder.DpiY,
                     pixels.DetachPixelData());

await encoder.FlushAsync(); // Write data to the stream

// Here you can use your stream
```

Load image in XAML

```
<Image Source="ms-appx:///Assets/Windows_10_Hero.png"/>
```

Your image is part of the application, in the Assets folder and marked as `Content`

```
<Image Source="ms-appdata:///local/Windows_10_Hero.png"/>
```

Your image was saved in your application's Local Folder

```
<Image Source="ms-appdata:///roaming/Windows_10_Hero.png"/>
```

Your image was saved in your application's Roaming Folder

Load image from Assets in Code

```
 ImageSource result = new BitmapImage(new Uri("ms-appx:///Assets/Windows_10_Hero.png"));
```

Use result to set the `Source` property of an `Image` control either though a `Binding` or code-behind

Load Image from StorageFile

```
public static async Task<ImageSource> FromStorageFile(StorageFile sf)
{
    using (var randomAccessStream = await sf.OpenAsync(FileAccessMode.Read))
    {
        var result = new BitmapImage();
        await result.SetSourceAsync(randomAccessStream);
        return result;
    }
}
```

Use result to set the `Source` property of an `Image` control either though a `Binding` or code-behind

Useful when you need to open images that are stored on the user's disk and not shipped with your application

Rendering a UI element to an Image

```
public static async Task<WriteableBitmap> RenderUIElement(UIElement element)
{
    var bitmap = new RenderTargetBitmap();
    await bitmap.RenderAsync(element);
    var pixelBuffer = await bitmap.GetPixelsAsync();
    var pixels = pixelBuffer.ToArray();
    var writeableBitmap = new WriteableBitmap(bitmap.PixelWidth, bitmap.PixelHeight);
    using (Stream stream = writeableBitmap.PixelBuffer.AsStream())
    {
        await stream.WriteAsync(pixels, 0, pixels.Length);
    }
    return writeableBitmap;
}
```

Since `WriteableBitmap` is an `ImageSource` you can use it to set the Source property of an Image control either though a Binding or code-behind

Save a WriteableBitmap to a Stream

```
public static async Task<IRandomAccessStream>
ConvertWriteableBitmapToRandomAccessStream(WriteableBitmap writeableBitmap)
{
    var stream = new InMemoryRandomAccessStream();

    BitmapEncoder encoder = await BitmapEncoder.CreateAsync(BitmapEncoder.JpegEncoderId,
stream);
    Stream pixelStream = writeableBitmap.PixelBuffer.AsStream();
    byte[] pixels = new byte[pixelStream.Length];
    await pixelStream.ReadAsync(pixels, 0, pixels.Length);

    encoder.SetPixelData(BitmapPixelFormat.Bgra8, BitmapAlphaMode.Ignore,
(uint)writeableBitmap.PixelWidth, (uint)writeableBitmap.PixelHeight, 96.0, 96.0, pixels);
    await encoder.FlushAsync();

    return stream;
}
```

Use the stream to save the Bitmap to a file.

Read Images online: https://riptutorial.com/uwp/topic/5170/images

Chapter 11: Images

Examples

Assigning a BitmapImage to Image's Source

```
Image img = new Image();
BitmapImage bitmap = new BitmapImage(new Uri("ms-appx:///Path-to-image-in-solution-directory",
UriKind.Absolute));
img.Source = bitmap;
```

Read Images online: https://riptutorial.com/uwp/topic/6564/images

Chapter 12: Navigation

Introduction

A soon as application have several pages/screens, a way of navigating among them is needed. With UWP applications, the navigation is handled by the [Frame][1] control. It displays [Page][2] instances, support the navigation to new pages and keep an history both for backward and forward navigation [1]: https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.frame.aspx [2]: https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.page.aspx

Examples

Create frame

A Frame is created like any other controls:

```
<Frame x:Name="contentRoot"
       Navigated="OnNavigated"
       Navigating="OnNavigating" />
```

The navigated/navigating events can then be intercepted to cancel the navigation or show/hide the back button.

```
private void OnNavigating(object sender, NavigatingCancelEventArgs e)
{
    if(contentRoot.SourcePageType  == e.SourcePageType && m_currentPageParameter ==
e.Parameter)
   {
       // we are navigating again to the current page, we cancel the navigation
       e.Cancel    = true;
   }
}

private void OnNavigated(object sender, NavigationEventArgs e)
{
    // user has navigated to a newest page, we check if we can go back and show the back
button if needed.
    // we can also alter the backstack navigation history if needed
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
(contentRoot.CanGoBack ? AppViewBackButtonVisibility.Visible :
AppViewBackButtonVisibility.Collapsed);
}
```

Navigate to a newest page

To navigate to a newest page, we can use the Navigate() method from the frame.

```
contentRoot.Navigate(typeof(MyPage), parameter);
```

where `contentRoot` is the Frame instance and `MyPage` a control inheriting from Page

In `MyPage`, the OnNavigatedTo() method will be called once the navigation will complete (ie when the user will enter the page) allowing us to triggering or finalizing the loading of the page data. The OnNavigatedFrom() method will be called when leaving the page allowing us to release what has to be released.

```
public class MyPage : Page
{
    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        // the page is now the current page of the application. We can finalized the loading
of the data to display
    }

    protected override void OnNavigatedFrom(NavigationEventArgs e)
    {
        // our page will be removed from the screen, we release what has to be released
    }
}
```

Confirming Navigation request using OnNavigatingFrom

```
private bool navigateFlag = false;

protected async override void OnNavigatingFrom(NavigatingCancelEventArgs e)
{
    base.OnNavigatingFrom(e);

    if (!navigateFlag)
        {
            e.Cancel = true;

            var dialog = new MessageDialog("Navigate away?", Confir,);
            dialog.Commands.Add(new UICommand("Yes", null, 0));
            dialog.Commands.Add(new UICommand("No", null, 1);

            dialog.CancelCommandIndex = 1;
            dialog.DefaultCommandIndex = 0;

            var result = await dialog.ShowAsync();

            if (Convert.ToInt16(result.Id) != 1)
            {
                navigateFlag= true;
                this.Frame.Navigate(e.SourcePageType);
            }

        }

    }
```

Read Navigation online: https://riptutorial.com/uwp/topic/8184/navigation

Chapter 13: Resources in UWP (StaticResource / ThemeResource) and ResourceDictionary

Introduction

In the new Windows 10 Applications there are many ways to reference a resource inside XAML code or in code behind. First of all you have to declare the resources in some accessible place. The easy way is to declare a `ResourceDictionary` in context, let's say in the current page.

Examples

1. Resource Dictionary

Snippet from MainPage.xaml

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

    <Page.Resources>
        <!-- Creates a resource dictionary in the page context -->
        <ResourceDictionary>
            <!-- This is a solid color brush resource
                 NOTE: Each resource inside a resource dictionary must have a key -->
            <SolidColorBrush x:Key="ColorRed">Red</SolidColorBrush>
        </ResourceDictionary>
    </Page.Resources>

    <!-- Using ThemeResource in here to access a resource already defined -->
    <Grid Background="{ThemeResource ColorRed}">

    </Grid>
</Page>
```

2. Global Resources

Resource dictionaries are accessible only inside the context they were declared, so if we intended to reference resources that are declared in one page context from another page they will not be found. So if we need global resources to be defined like the ones that comes with the framework we do it in App.xaml

Snippet from App.xaml

```
<Application
    x:Class="MyNewApp.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    RequestedTheme="Dark">

    <Application.Resources>
        <ResourceDictionary>
            <SolidColorBrush x:Key="ColorRed">Red</SolidColorBrush>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

This way we can access `ColorRed` color resource from anywere in our app. But wait, we don't want to infest that little file with all our app's resources! So we do `MergedDictionaries`

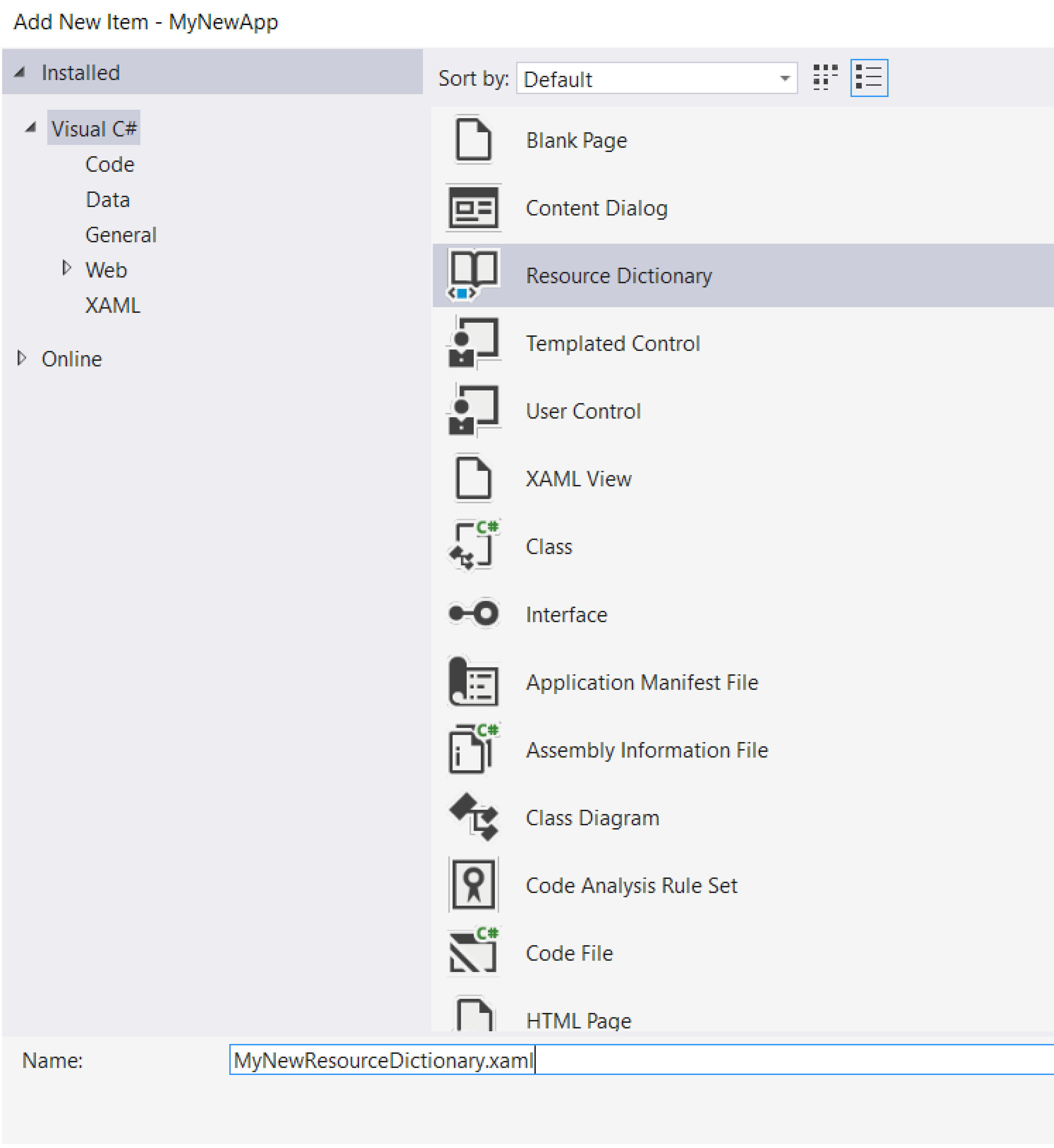
3. Merged Dictionaries

Almost usually things are a little bit more complex and to support scalability we should split things apart. So we can define various files containing different resources dictionaries, i.e. resources for UI controls' themes, resources for texts and so on, then we merge them all together in App.xaml file.

Snippet from App.xaml

```
<Application
    x:Class="MyNewApp.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    RequestedTheme="Dark">

    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="/Assets/Themes/GeneralStyles.xaml"/>
                <ResourceDictionary Source="/Assets/Themes/TemplatedControls.xaml"/>
                <ResourceDictionary Source="/Assets/Strings/Texts.xaml"/>
                <ResourceDictionary Source="/Assets/Strings/ErrorTexts.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

You can create a new dictionary file by right clicking on Asset folder [Add -> New Item]


Add New Item - MyNewApp

Name: MyNewResourceDictionary.xaml

4. Accessing Resources

We now need to access to our declared resources, in order to do that from XAML code we use `{ThemeResource ResourceKey}` or `{StaticResource ResourceKey}`

```
to be continued later.
```

Read Resources in UWP (StaticResource / ThemeResource) and ResourceDictionary online: https://riptutorial.com/uwp/topic/10511/resources-in-uwp--staticresource---themeresource--and-resourcedictionary

Chapter 14: Settings and app data

Examples

Store and retrieve settings

UWP applications can easily store simple settings in a key/value store locally or even in the cloud so your application or a game can share settings between different user's devices.

Following data types can be used for settings:

- UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double
- Boolean
- Char16, String
- DateTime, TimeSpan
- GUID, Point, Size, Rect

Start by retrieving the local and/or roaming data container.

```
Windows.Storage.ApplicationDataContainer localSettings =
Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.ApplicationDataContainer roamingSettings =
Windows.Storage.ApplicationData.Current.RoamingSettings;
```

To create or write a setting, use ApplicationDataContainer.Values property to access the settings in the data container. For example lets create a local setting named `FontSize` with an `int` value `10` and roaming setting `Username` with a `string` value `Bob`.

```
localSettings.Values["FontSize"] = 10;
roamingSettings.Values["Username"] = "Bob";
```

To retrieve the setting, use the same ApplicationDataContainer.Values property that you used to create the setting.

```
int fontSize = localSettings["FontSize"];
string username = roamingSettings["Username"];
```

Good practice is to check if a setting exists before retrieving it.

```
if (localSettings.Values.ContainsKey("FontSize"))
    int fontSize = localSettings["FontSize"];

if (roamingSettings.Values.ContainsKey("Username"))
    string username = roamingSettings["Username"];
```

Roaming settings have size quota. Use RoamingStorageQuota property go get it.

You can find more about settings, their limits and code examples on MSDN.

Save data to application cache

The ApplicationData.Current.LocalFolder api allows us to get access to the application cache :

```
var file = await ApplicationData.Current.LocalFolder.CreateFileAsync("myFile.dat",
CreationCollisionOption.ReplaceExisting);
```

The FileIO class contains a set of utility methods to easily add data to a file :

```
await FileIO.WriteBytesAsync(file, array);
await FileIO.AppendTextAsync(file, "text");
await FileIO.WriteBufferAsync(file, iBuffer);
```

Read Settings and app data online: https://riptutorial.com/uwp/topic/5944/settings-and-app-data

Chapter 15: Theme Resources

Syntax

- C# : Application.Current.Resources["yourColorKey"]
- Xaml : {ThemeResource yourColorKey}

Parameters

Parameter	Purpose
`yourColorKey`	A key you give to get a `Color` object back. It differs between C# and Xaml

Remarks

UWP allows you to take full control of the advantages of Windows 10. Some of these advantages are graphical, as the Accent color or Dark/Light themes.

To prepare your app to be compatible with these feature, a bunch of premade colors have been implemented in UWP to change with the Accent color of the OS the program runs on, or with the theme choice of the user.

There are two "ways" of doing this :

- Diretly in Xaml, using the `Color = {ThemeResource x}` Attribute (or whatever attribute that takes a `Brush`as value, like BorderBrush, Background, etc.)

- In C# Code Behind, by Searching for the color in the Resource directory of the current app. This gives a `Color` object, so if you want to put it in the `Color` property of an object you referenced from your Xaml, you'll need to make a new brush like this :

  ```
  new SolidColorBrush(Application.Current.Resources["yourColorKey"])
  ```

For a reference of color keys in c#, please consult :

> https://msdn.microsoft.com/windows/uwp/controls-and-patterns/xaml-theme-resources

Examples

Access to Theme Resources in Xaml

Snippet from MyExampleFile.xaml

```
<TextBlock Foreground="{ThemeResource SystemControlBackgroundAccentBrush}"
         Text="This is a colored textbox that use the Accent color of your Windows 10"/>

<TextBlock Foreground="{ThemeResource SystemControlBackgroundBaseHighBrush}"
         Text="This is a colored textbox that use a color that is readable in both Light and
Dark theme"/>
```

Access to Theme Resources in C#

Snippet from MyExampleFile.xaml

```
<TextBlock x:Name="MyTextBlock"
         Text="This is a TextBlock colored from the code behind"/>
```

Snippet from MyExampleFile.xaml.cs

```
    // We use the application's Resource dictionary to get the current Accent of your Windows
10
    MyTextBlock.Color = new
SolidColorBrush(Application.Current.Resources["SystemAccentColor"]);
```

Read Theme Resources online: https://riptutorial.com/uwp/topic/7527/theme-resources

Chapter 16: Unit Testing for UWP

Introduction

I would like to show you how to create Unit Tests for Universal Windows 10 Application. To test UWP apps we will use xUnit.net Framework about which you can read more from the link I provided in remarks section.

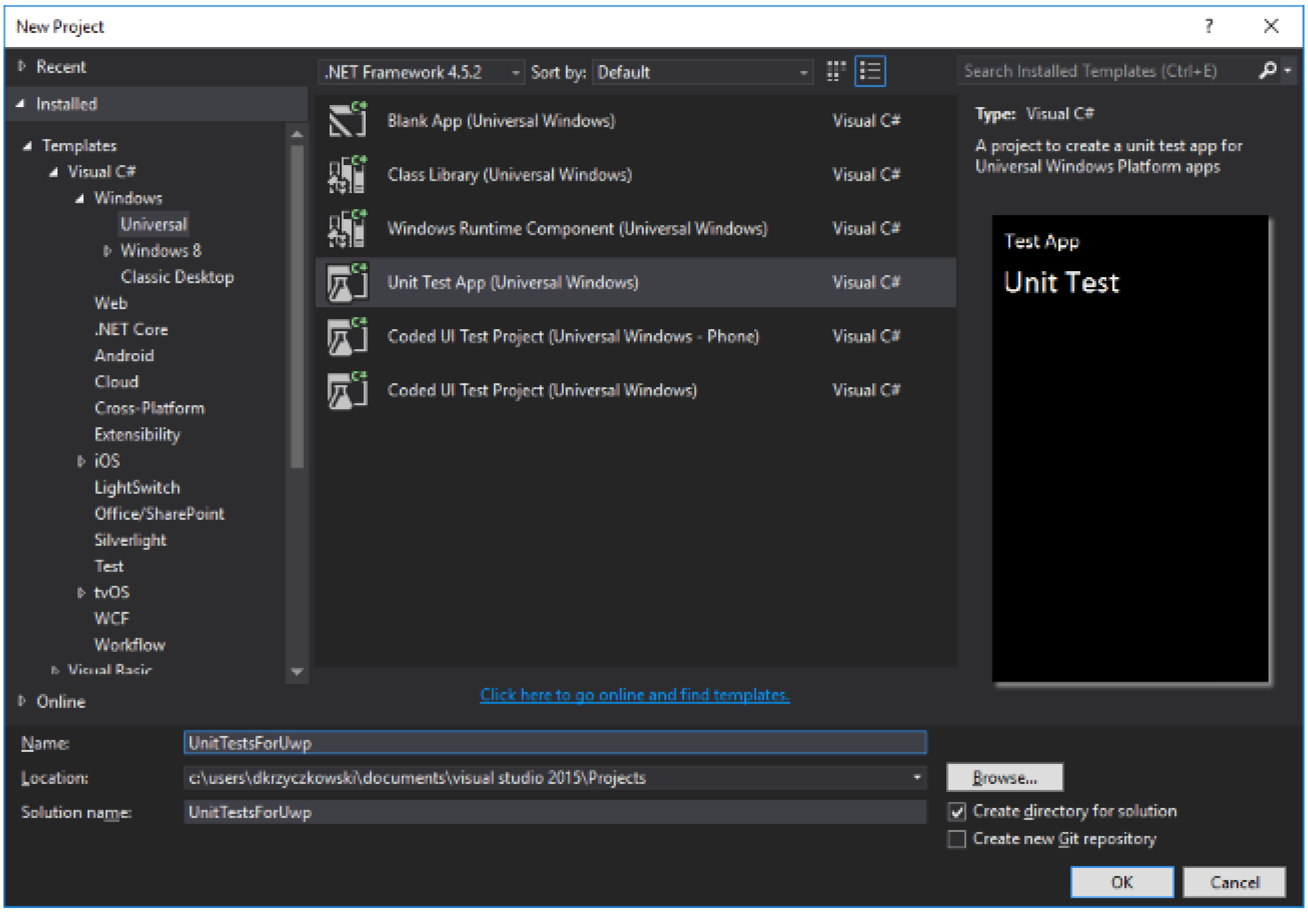
Remarks

You can read more about xUnit Framewwork: https://xunit.github.io/docs/getting-started-uwp.html

Examples

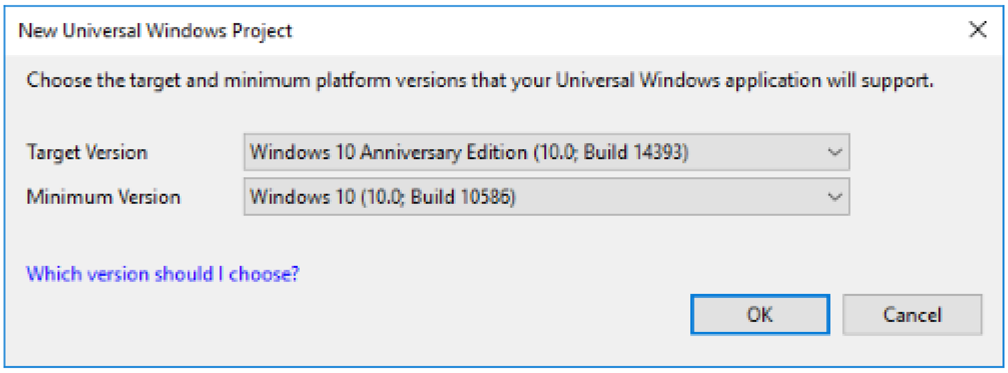
Configure Test Application

Once you have your UWP application ready for tests you should add test application to your solution. To do it "right" click on the solution and choose "Unit Test App (Universal Windows)":


New Project
Recent
Installed
Templates
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Web
.NET Core
Android
Cloud
Cross-Platform
Extensibility
iOS
LightSwitch
Office/SharePoint
Silverlight
Test
tvOS
WCF
Workflow
Visual Basic
Online
.NET Framework 4.5.2
Sort by: Default
Search Installed Templates (Ctrl+E)
Blank App (Universal Windows) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Universal Windows - Phone) Visual C#
Coded UI Test Project (Universal Windows) Visual C#
Type: Visual C#
A project to create a unit test app for Universal Windows Platform apps
Test App
Unit Test
Click here to go online and find templates.
Browse...
OK
Cancel

Name: UnitTestsForUwp
Location: c:\users\dkrzyczkowski\documents\visual studio 2015\Projects
Solution name: UnitTestsForUwp
[✓] Create directory for solution
[ ] Create new Git repository

Once you add it to the solution there are few more steps required to configure it. You will be asked for selecting target and minimum platform version:


New Universal Windows Project

Choose the target and minimum platform versions that your Universal Windows application will support.

Target Version: Windows 10 Anniversary Edition (10.0; Build 14393)

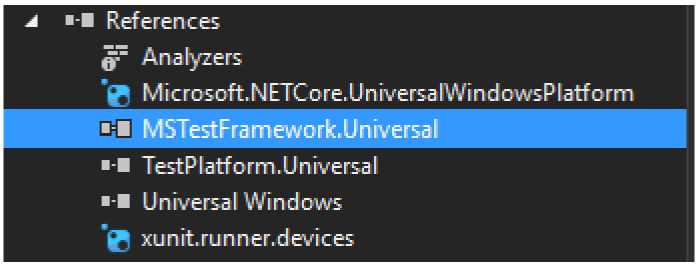
Minimum Version: Windows 10 (10.0; Build 10586)

Once you select them, open "project.json" file and add below dependencies:

```
"dependencies":
 {
  "Microsoft.NETCore.UniversalWindowsPlatform": "5.1.0",
  "xunit.runner.visualstudio": "2.1.0",
  "xunit": "2.1.0",
  "xunit.runner.devices": "2.1.0"
 }
```

These are used to download and add NuGet xUnit Framework packages to make unit tests easy for UWP application.

Remove reference called “MSTestFramework.Universal”:



Now open “UnitTest.cs” file. Modify it to look like below:

```
using System;
using Xunit;

namespace UnitTestsForUwp
{
  public class UnitTest1
   {
       [Fact]
       public void TestMethod1()
        {
          Assert.Equal(4, 4);
        }

       [Theory]
       [InlineData(6)]
       public void TestMethod2(int value)
        {
          Assert.True(IsOdd(value));
        }
       bool IsOdd(int value)
        {
          return value % 2 == 1;
        }
     }
   }
}
```

It is good to stop here for a moment to talk a little bit about xUnit attributes:

a. Fact- tests which are always true. They test invariant conditions.

b. Theory – tests which are only true for a particular set of data.

Now we would like to prepare the app to display information about tests but not only - it is good to have one good way to start tests. To achieve that we need to make small changes in "UnitTestApp.xaml" file. Open it and replace all code with pasted below:

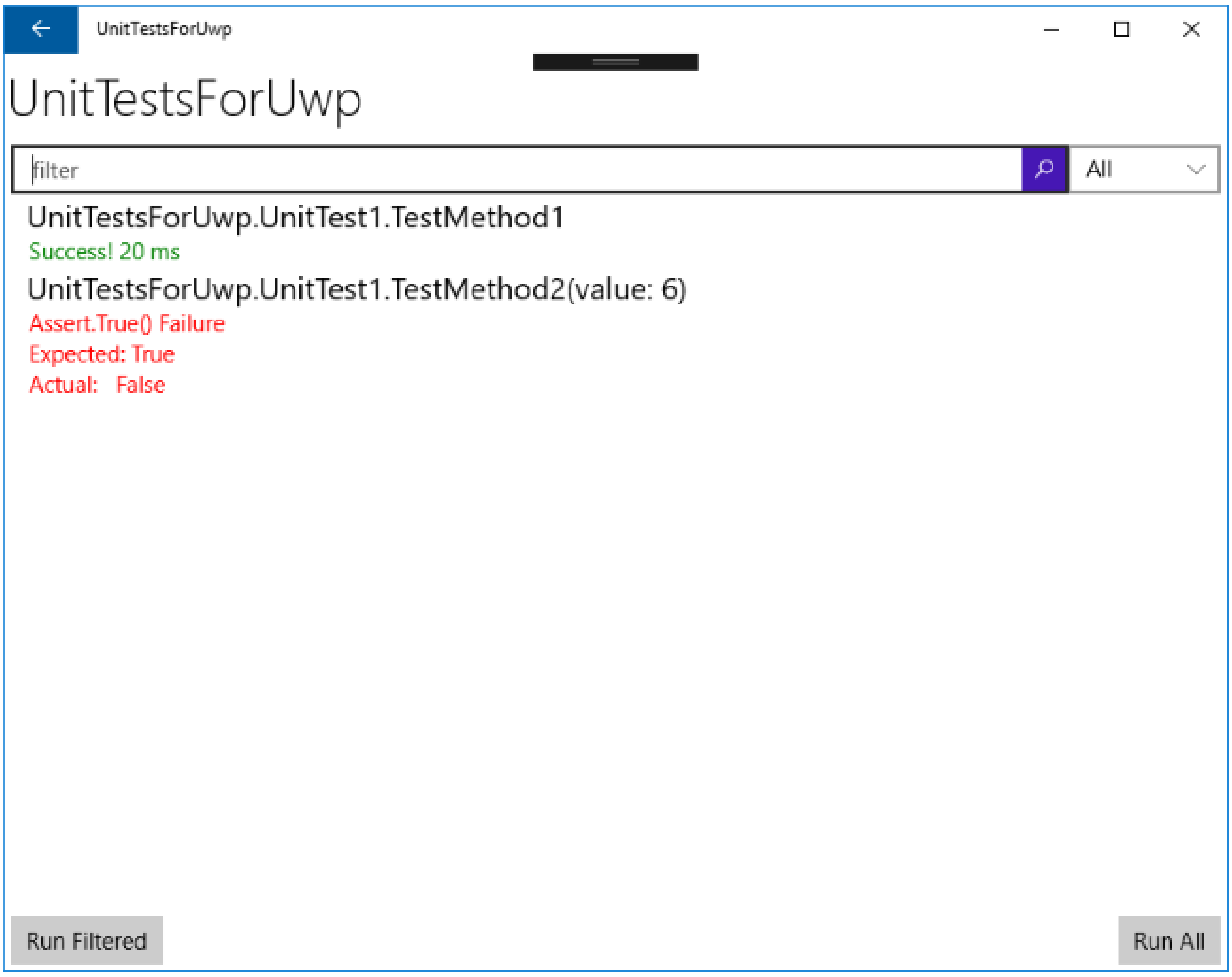
```
<ui:RunnerApplication
 x:Class="UnitTestsForUwp.App"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:local="using:UnitTestsForUwp"
 xmlns:ui = "using:Xunit.Runners.UI"
 RequestedTheme="Light">
</ui:RunnerApplication>
```

Remember that "local" should have the same name like your namespace.

Now open "UnitTestApp.xaml.cs" and replace code with below:

```
sealed partial class App : RunnerApplication
 {
   protected override void OnInitializeRunner()
    {
      AddTestAssembly(GetType().GetTypeInfo().Assembly);
      InitializeRunner();
    }
    partial void InitializeRunner();
  }
```

That's it! Now rebuild project and launch test application. As you can see below you have access to all your tests, you can start them and check results:

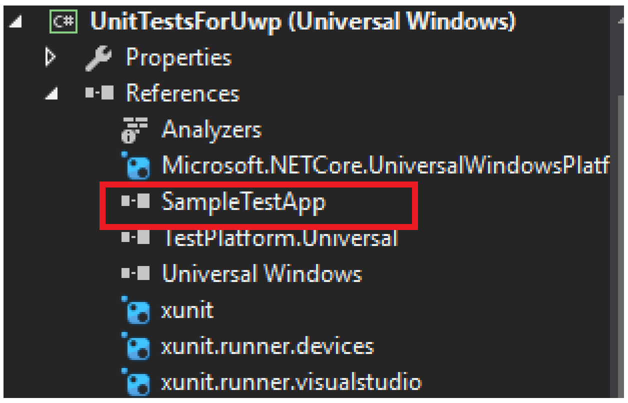

UnitTestsForUwp

UnitTestsForUwp.UnitTest1.TestMethod1
Success! 20 ms
UnitTestsForUwp.UnitTest1.TestMethod2(value: 6)
Assert.True() Failure
Expected: True
Actual: False

Connect Test Application with target app code

Once your test application is ready you can connect it with code for which you want to write unit tests.

Either you have you code in PCL, or in UWP app project (I assume that you applied MVVM pattern) just add reference to it in Test Application project:


UnitTestsForUwp (Universal Windows)
- Properties
- References
  - Analyzers
  - Microsoft.NETCore.UniversalWindowsPlatf
  - SampleTestApp
  - TestPlatform.Universal
  - Universal Windows
  - xunit
  - xunit.runner.devices
  - xunit.runner.visualstudio

Now you have access to all your code from Test Application. Create Unit Tests you want. Just use "Facts" or "Theories".

Mock some functionality

Once you have everything prepared to write your Unit Tests it is worth to mention about mocking. There is new framework called "SimpleStubs" which enables you to create mocks based on the interfaces.

Simple case from GitHub documentation:

```
//Sample interface:
public interface IPhoneBook
 {
   long GetContactPhoneNumber(string firstName, string lastName);
   long MyNumber { get; set; }
   event EventHandler<long> PhoneNumberChanged;
 }

//Mocked interface:
var stub = new StubIPhoneBook().GetContactPhoneNumber((firstName, lastName) => 6041234567);
```

You can read more about it here: https://github.com/Microsoft/SimpleStubs

Read Unit Testing for UWP online: https://riptutorial.com/uwp/topic/8634/unit-testing-for-uwp

Chapter 17: Using JavaScript in WebView

Introduction

This document shows you how you can use JavaScript in WebView.

This document cover: Getting HTML from the WebView, Entering text in the text box on the website, Simulate click to click a website button

Syntax

- `await webView.InvokeScriptAsync("eval", new string[] { functionString })` - to use JavaScript
- `.documentElement` - to get a reference to the root node of the document
- `.getElementsByClassName(Class_Name)` - to get elements usign Class Name
- `.getElementsByTagName(Tab_Name)` - to get elements using Tag Name
- `.getElementById(ID)` - to get element using ID
- `.nodeName` - to get the node name
- `.childNodes` - to get the child elements
- `.outerHTML` - to Get the Outer HTML
- `.innerHTML` - to Get the Inner HTML
- `.innerText` - to Get or Set InnerText
- `.click()` - to Simulate click

Remarks

Here is a Sample app to LogIn to StackOverFlow

Examples

Getting HTML from the WebView

Use `.outerHTML` to get the HTML

Here is a code sample to get the entire HTML of the website

```
private async void GetHTMLAsync()
{
    var siteHtML = await webView.InvokeScriptAsync("eval", new string[] {
"document.documentElement.outerHTML;" });
}
```

Entering text in the text box on the website

Use `.innerText` to set the value

Here is a code sample to enter text in Search Box on Bing website

```
private async void EnterTextAsync(string enterText)
{
    var functionString =
string.Format(@"document.getElementsByClassName('b_searchbox')[0].innerText = '{0}';",
enterText);
    await webView.InvokeScriptAsync("eval", new string[] { functionString });
}
```

Simulate click to click a website button

Use `.click()` to simulate click

Here is a code sample to click search button on Bing website

```
private async void SimulateClickAsync()
{
    var functionString =
string.Format(@"document.getElementsByClassName('b_searchboxSubmit')[0].click();");
    await webView.InvokeScriptAsync("eval", new string[] { functionString });
}
```

Read Using JavaScript in WebView online: https://riptutorial.com/uwp/topic/10794/using-javascript-in-webview

Chapter 18: UWP background tasks

Remarks

- For registering a background task that runs in a seperate process, you have to go to the "Declarations" Tab in the Package.appxmanifest and add a new "Background Task" and set the entry point.
- Registering a single-process background task can be done by means of `BackgroundTaskBuilder`, but the application will throw an exception if you register a task twice, so you must check if you have already registered a task.
- The app must gain authority to register a new task, this can be done by calling `BackgroundExecutionManager.RequestAccessAsync()`, but make sure that you really have the permission. The call returns the type of access (`BackgroundAccessStatus` enum) which will indicate whether you have access or not.
- Tasks registered are kept until the package is uninstalled, but it won't hurt to check the tasks you need on every launch, bug happens!
- When the application is updated, permission to register a new task is revoked. To keep your app running after an update, especially if you have added a new task register, you have to *remove and request the access over*, by means of `BackgroundAccessManager`. One method to know if your app is updated, is to register another task with a `SystemTrigger`, type of `SystemTriggerType.ServicingComplete`.

Examples

Registering a Task

```
/// <summary>
/// Registers a background task in the system waiting to trigger
/// </summary>
/// <param name="taskName">Name of the task. Has to be unique</param>
/// <param name="taskEntryPoint">Entry point (Namespace) of the class (has to implement
IBackgroundTask and has to be in a Windows Runtime Component) to start</param>
/// <param name="trigger">What has to be triggered to start the task</param>
/// <param name="condition">Optional condition. Can be null</param>
/// <param name="recreateIfExists">Should the Task be recreated if it already exists?</param>
/// <returns></returns>
public BackgroundTaskRegistration RegisterTask(string taskName, string taskEntryPoint,
IBackgroundTrigger trigger, IBackgroundCondition condition = null) {
    Debug.WriteLine("Try registering task: " + taskName);

    var builder = new BackgroundTaskBuilder {
        Name = taskName,
        TaskEntryPoint = taskEntryPoint
    };

    builder.SetTrigger(trigger);

    if (condition != null) {
        builder.AddCondition(condition);
    }

    try {
        var task = builder.Register();
        Debug.WriteLine("Task successfully registered");
        return task;
    } catch (Exception exception) {
        Debug.WriteLine("Error creating Task: " + exception);
        return null;
    }
}
```

Get a registered task by its name

```
/// <summary>
/// Gets a BackgroundTask by its name
/// </summary>
/// <param name="taskName">Name of the task to find</param>
/// <returns>The found Task or null if none found</returns>
public BackgroundTaskRegistration TaskByName(string taskName) =>
       BackgroundTaskRegistration.AllTasks.FirstOrDefault(x =>
x.Value.Name.Equals(taskName)).Value as BackgroundTaskRegistration;
```

The task

```
public sealed class BackgroundTask : IBackgroundTask {

    private BackgroundTaskDeferral _deferral;

    /// <summary>
    /// Registers the listener to check if the button is pressed
    /// </summary>
    /// <param name="taskInstance">An interface to an instance of the background task. The
system creates this instance when the task has been triggered to run.</param>
    public async void Run(IBackgroundTaskInstance taskInstance) {
        _deferral = taskInstance.GetDeferral();

        //Do async operations here

        _deferral.Complete();
    }
}
```

Check if Task is registered

```
private bool IsTaskRegistered(string taskName) =>
        BackgroundTaskRegistration.AllTasks.Any(x => x.Value.Name.Equals(taskName));
```

Triggering a task manually

```
var trigger = new ApplicationTrigger();
TaskHandlerMentionedInThisTutorial.RegisterTask(TaskName, entryPoint, trigger, null, true);

await trigger.RequestAsync();
```

Unregistering a task

```
/// <summary>
/// Unregister a single background task with given name
/// </summary>
/// <param name="taskName">task name</param>
/// <param name="cancelTask">true if task should be cancelled, false if allowed to
finish</param>
public void UnregisterTask(string taskName, bool cancelTask) =>
    BackgroundTaskRegistration.AllTasks.First(x =>
x.Value.Name.Equals(taskName)).Value?.Unregister(cancelTask);

/// <summary>
/// Unregister an active group of background tasks, which name contains given string
/// </summary>
/// <param name="taskNamePart">part of the task name</param>
/// <param name="cancelTask">true if tasks should be cancelled, false if allowed to
finish</param>
public void UnregisterTasks(string taskNamePart, bool cancelTask)
{
    foreach (var task in BackgroundTaskRegistration.AllTasks.Where(x =>
x.Value.Name.Contains(taskNamePart)))
        task.Value.Unregister(cancelTask);
}
```

Register background task with trigger

The background task are a great way to perform some work while your application is not running. Before being able to use then , you will have to register them.

Here is a sample of a background task class including the registration with a trigger and a condition and the Run implementation

```
public sealed class Agent : IBackgroundTask
{
    public void Run(IBackgroundTaskInstance taskInstance)
    {
        // run the background task code
    }

    // call it when your application will start.
    // it will register the task if not already done
    private static IBackgroundTaskRegistration Register()
    {
        // get the entry point of the task. I'm reusing this as the task name in order to get
an unique name
        var taskEntryPoint   = typeof(Agent).FullName;
        var taskName         = taskEntryPoint;

        // if the task is already registered, there is no need to register it again
        var registration            = BackgroundTaskRegistration.AllTasks.Select(x =>
x.Value).FirstOrDefault(x => x.Name == taskName);
        if(registration != null) return registration;

        // register the task to run every 30 minutes if an internet connection is available
        var taskBuilder             = new BackgroundTaskBuilder();
        taskBuilder.Name            = taskName;
        taskBuilder.TaskEntryPoint  = taskEntryPoint;
        taskBuilder.SetTrigger(new TimeTrigger(30, false));
        taskBuilder.AddCondition(new SystemCondition(SystemConditionType.InternetAvailable));

        return taskBuilder.Register();
    }
}
```

Read UWP background tasks online: https://riptutorial.com/uwp/topic/2494/uwp-background-tasks

Chapter 19: UWP Hello World

Syntax

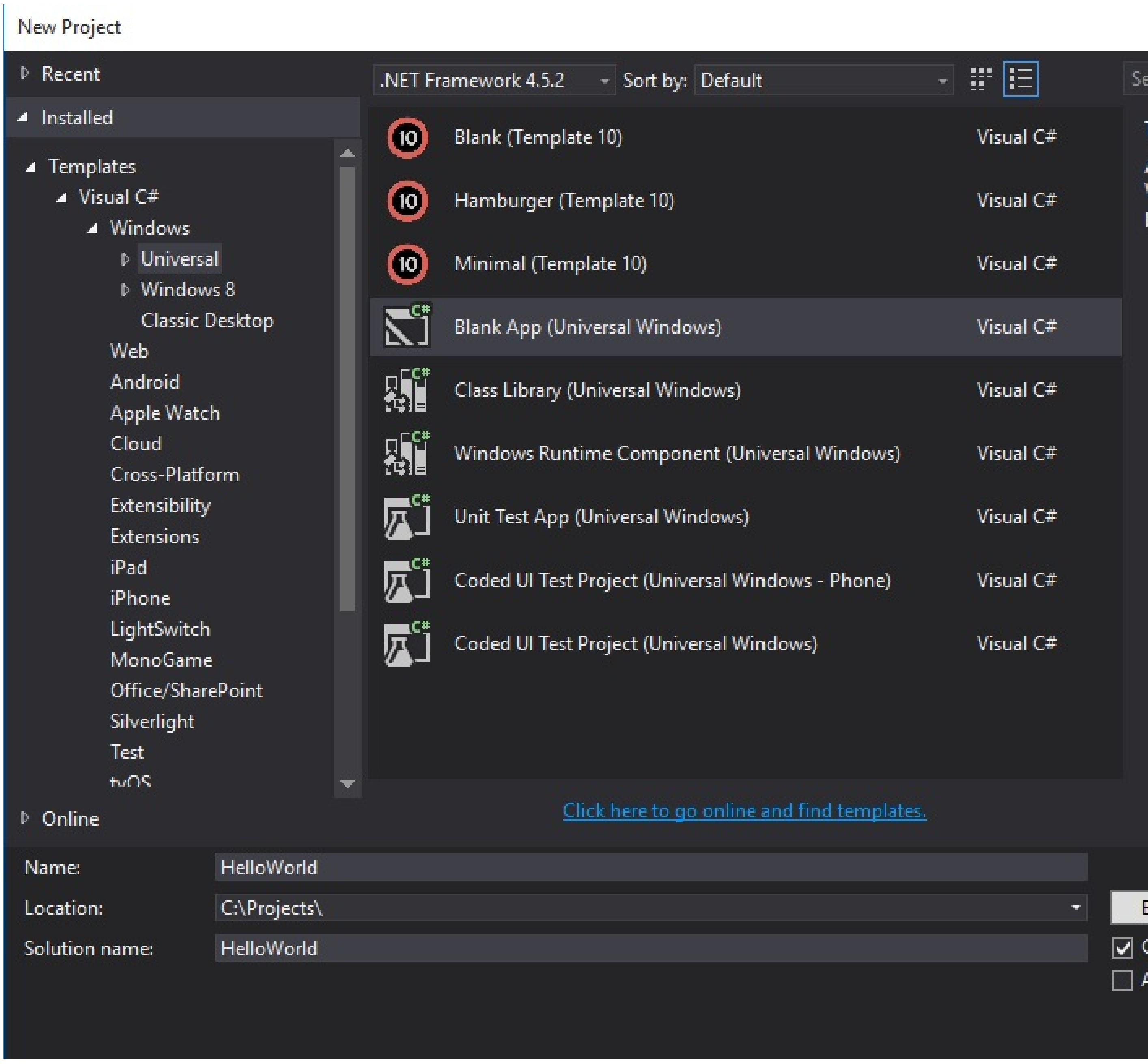
- This is the simple example of popular "Hello World!" for Universal Windows Platform on Windows 10.

Examples

Hello World - Universal Windows Platform

After launching Visual Studio 2015, go to `File → New → Project`. In the *New Project* dialog box, browse in the templates tree to `Visual C# → Windows → Universal` and select `Blank App (Universal Windows)`. Next, we need to fill the form to describe the Application:

1. **Name**: this is the name of the application which will be displayed to the user. Set it to `HelloWorld` or use a custom title.
2. **Location**: indicates where the project will be stored
3. **Solution Name**: this is a kind of container of projects which groups several projects related to the same application (for example a solution could be composed of a UI project and a model project). You can put the same `Name` as your initial project.



New Project

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows
        - Universal
        - Windows 8
        - Classic Desktop
      - Web
      - Android
      - Apple Watch
      - Cloud
      - Cross-Platform
      - Extensibility
      - Extensions
      - iPad
      - iPhone
      - LightSwitch
      - MonoGame
      - Office/SharePoint
      - Silverlight
      - Test
      - tvOS
- Online

.NET Framework 4.5.2 Sort by: Default

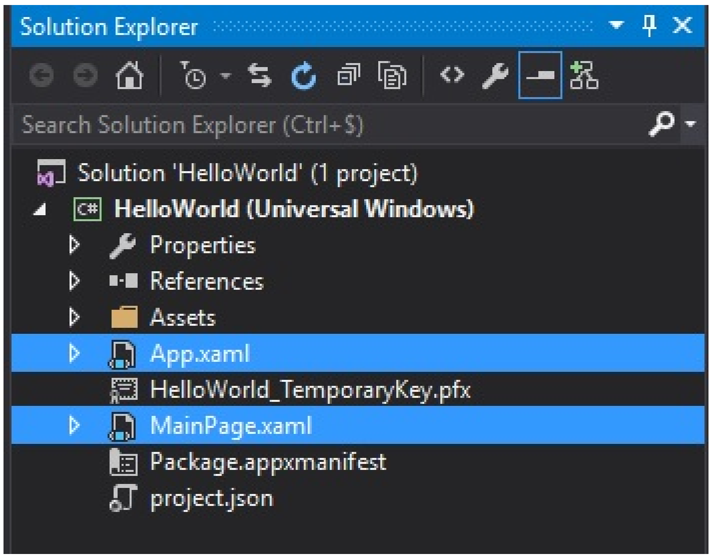
Blank (Template 10)	Visual C#
Hamburger (Template 10)	Visual C#
Minimal (Template 10)	Visual C#
Blank App (Universal Windows)	Visual C#
Class Library (Universal Windows)	Visual C#
Windows Runtime Component (Universal Windows)	Visual C#
Unit Test App (Universal Windows)	Visual C#
Coded UI Test Project (Universal Windows - Phone)	Visual C#
Coded UI Test Project (Universal Windows)	Visual C#

Click here to go online and find templates.

Name:	HelloWorld
Location:	C:\Projects\
Solution name:	HelloWorld

Content of the default project

You will obtain a project with the following files:


Solution Explorer
Search Solution Explorer (Ctrl+$)
Solution 'HelloWorld' (1 project)
HelloWorld (Universal Windows)
Properties
References
Assets
App.xaml
HelloWorld_TemporaryKey.pfx
MainPage.xaml
Package.appxmanifest
project.json

1. **Package.appxmanifest**: describes properties of your application. It contains some UI settings such as its disaply name, its logo, the supported rotations. And it also contains technical settings such as the entry point of the application (wich is the `App` class by default). Finally, it also list authorizations that are required by your application in the *Capabilities* tab; for example if your want to use the webcam in your application you will have to check the corresponding capabilities.
2. **App.xaml / App.xaml.cs**: the `App` class is the default entry point of your application. The xaml files can hold resources shared across the whole application such as styles setting or instance of a class that you want to share such as a ViewModel locator. The code-behind files contains all the startup code of the application. By default, it implements the `OnLaunched` method which is invoked by the end user. It initializes the window and navigate to the first page of the application (by default the `MainPage` class).
3. **MainPage.xaml / MainPage.xaml.cs**: this is the initial page of our application. It contains only an empty Grid which is a layout control.

Modify the view

Open the `MainPage.xaml` and replace the Grid control with

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="auto" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Button Click="Button_Click">Say Hello !</Button>
    <TextBlock Grid.Column="1"
               VerticalAlignment="Center"
               x:Name="myText"
               Text="Click the button." />
</Grid>
```

This will create a grid with two columns. The first column as a width set to `auto` which means that it will automatically be set in function of the size of its children. The second column will stretch to fill the remaining space in the window. This grid contains two elements:

- a `Button` that lies in the first column. The click event is bind to the method `Button_Click` on the code-behind and its caption Text is *"Say Hello!"*.
- a `TextBlock` that lies in the second column. It's text is set to *"Click the button."*. And we have set a name to this control with the help of the attribute `x:Name`. This is required to be able to use the control in the code-behind. In the `MainPage.xaml.cs`, add the following code:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    this.myText.Text = "Hello World!";
}
```

This is the method that will be called when the user clicks (or taps) the button. And it will updates the `TextBlock` and set its text to *"Hello World!"*.

Running the application

To run the application, you can use the menu `Debug → Start Debugging` or the shortcut `F5`. By default, it will run the application on your Local Machine.

Read UWP Hello World online: https://riptutorial.com/uwp/topic/2339/uwp-hello-world

Chapter 20: WebView

Examples

Add a WebView to the UI

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <WebView x:Name="MyWebView" />
</Grid>
```

Open a website

```
MyWebView.Navigate(new Uri("http://www.url.com"));
```

Open local html page

```
MyWebView.Navigate(new Uri("ms-appdata:///local/Downloads/index.html"));
```

Read WebView online: https://riptutorial.com/uwp/topic/6541/webview

Chapter 21: WebView navigation

Remarks

All examples that fetch data from a remote URL, has to have "Internet (client)" capability checked in the Package.appxmanifest. For examples that only manipulate local data it's not necessary.

Examples

Navigate to Uri

This code simply navigates WebView to some Uri:

```
this.webView.Navigate(new Uri("http://stackoverflow.com/"));
```

or

```
this.webView.Source = new Uri("http://stackoverflow.com/");
```

Navigate with HttpRequestMessage

Set custom user agent and navigate to Uri:

```
var userAgent = "my custom user agent";
var uri = new Uri("http://useragentstring.com/");
var requestMessage = new HttpRequestMessage(HttpMethod.Get, uri);
requestMessage.Headers.Add("User-Agent", userAgent);

this.webView.NavigateWithHttpRequestMessage(requestMessage);
```

Navigate to string

Show specified html string in WebView:

```
var htmlString =
    @"<!DOCTYPE html>
      <html>
          <head><title>HTML document</title></head>
          <body>
              <p>This is simple HTML content.</p>
          </body>
      </html>";

this.webView.NavigateToString(htmlString);
```

Open HTML file from app package

You can easily open a file from your app package, but Uri scheme must be "ms-appx-web" instead of "ms-appx":

```
var uri = new Uri("ms-appx-web:///Assets/Html/html-sample.html");
this.webView.Navigate(uri);
```

Open HTML file from app local folder or temp folder

To open a file from local folder or temp folder, target file **must not** be located in those folders' root. For security reasons, to prevent other content from being exposed by WebView, the file meant for displaying must be located in a subfolder:

```
var uri = new Uri("ms-appdata:///local/html/html-sample.html");
this.webView.Navigate(uri);
```

NavigateToLocalStreamUri

In case when NavigateToString can't handle some content, use NavigateToLocalStreamUri method. It will force every locally-referenced URI inside the HTML page to call to the special resolver class, which can provide right content on the fly.

Assets/Html/html-sample.html file:

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML document</title>
    </head>
    <body>
        <p>This is simple HTML content.</p>
        <img src="cat.jpg"/>
    </body>
</html>
```

Code:

```
protected override void OnNavigatedTo(NavigationEventArgs args)
{
    // The Uri resolver takes is in the form of "ms-local-stream://appname_KEY/folder/file"
    // For simplicity, there is method BuildLocalStreamUri which returns correct Uri.
    var uri = this.webView.BuildLocalStreamUri("SomeTag", "/html-sample.html");
    var resolver = new StreamUriResolver();
    this.webView.NavigateToLocalStreamUri(uri, resolver);

    base.OnNavigatedTo(args);
}


public sealed class StreamUriResolver : IUriToStreamResolver
{
    public IAsyncOperation<IInputStream> UriToStreamAsync(Uri uri)
    {
        if (uri == null)
        {
            throw new ArgumentNullException(nameof(uri));
        }

        var path = uri.AbsolutePath;

        return GetContent(path).AsAsyncOperation();
    }


    private async Task<IInputStream> GetContent(string uriPath)
    {
        Uri localUri;

        if (Path.GetExtension(uriPath).Equals(".html"))
        {
            localUri = new Uri("ms-appx:///Assets/Html" + uriPath);
        }
        else
        {
            localUri = new Uri("ms-appdata:///local/content" + uriPath);
        }

        var file = await StorageFile.GetFileFromApplicationUriAsync(localUri);
        var stream = await file.OpenAsync(FileAccessMode.Read);

        return stream.GetInputStreamAt(0);
    }
}
```

This code will take HTML page from app package and embed content from local folder into it. Provided that you have image "cat.jpg" in /local/content folder, it will show HTML page with cat image.

Read WebView navigation online: https://riptutorial.com/uwp/topic/6321/webview-navigation

Chapter 22: Working with Filesystem

Examples

How to share data across multiple devices in Win10 UWP App

To make an app more cohesive, we often need to keep user's personal settings and preferences consistent across multiple devices that have been logged in with one Microsoft account. In this sample, we use roaming data to store and to load UI settings, game process and user info. But the roaming data has its own limit: we cannot store large file in the roaming folder. The system suspends data replication for all apps in the package to the cloud until the current size no longer exceeds the maximum size. Therefore, in this sample, we haven't stored the user image in the roaming folder. Instead, it is stored in the local folder.

```
private async void LoadRoamingData()
{
    //Get background color
    object color = roamingSettings.Values["BackgroundColor"];
    if (color != null)
    {
        if (ViewModel.ColorList.Keys.Contains(color.ToString()))
        {
            Color backgroundColor = ViewModel.ColorList[color.ToString()];
            ViewModel.BackgroundColor = new SolidColorBrush(backgroundColor);
            comboBackgroundColor.SelectedValue = color.ToString();
        }
    }
    //Get game process stored in the roaming file
    try
    {
        StorageFile processFile = await roamingFolder.GetFileAsync(processFileName);
        string process = await FileIO.ReadTextAsync(processFile);
        int gameProcess;
        if (process != null && int.TryParse(process.ToString(), out gameProcess) &&
gameProcess > 0)
        {
            ViewModel.GameProcess = gameProcess;
        }
    }
    catch { }

    //Get user name
    object userName = roamingSettings.Values["UserName"];
    if (userName != null && !string.IsNullOrWhiteSpace(userName.ToString()))
    {
        ViewModel.UserName = userName.ToString();
    }
}
```

For more information, see https://code.msdn.microsoft.com/How-to-share-data-across-d492cc0b.

Read Working with Filesystem online: https://riptutorial.com/uwp/topic/6480/working-with-filesystem

Credits

S. No	Chapters	Contributors
1	Getting started with uwp	Community, dub stylee, Jerin, Josh, Pratyay
2	Adaptive UI	Vincent
3	Application Lifecycle	Daniel Krzyczkowski
4	Binding vs x:Bind	Alias Varghese, Anthony Russell, Askerad, Raben des Unbekannten, Vincent
5	Convert image size and crop image file in Windows Universal app	Dev-Systematix
6	Device Families	Josh, M. Pipal, Vincent
7	File name qualifiers	Raben des Unbekannten
8	How to get current DateTime in C++ UWP	Mo0gles
9	Images	acedened, AlexDrenea, Bart
10	Navigation	Takarii, Vincent
11	Resources in UWP (StaticResource / ThemeResource) and ResourceDictionary	Ivan Carmenates García
12	Settings and app data	khamitimur, Vincent
13	Theme Resources	Askerad, Bart, Raamakrishnan A.
14	Unit Testing for UWP	Daniel Krzyczkowski
15	Using JavaScript in WebView	Vijay Nirmal

16	UWP background tasks	acedened, Askerad, Raben des Unbekannten, RamenChef, Romasz, TableCreek, Vincent
17	UWP Hello World	Almir Vuk, Arnaud Develay, Josh, user2950509
18	WebView	TableCreek
19	WebView navigation	Andrey Ashikhmin
20	Working with Filesystem	Dale Chen