# Windows 10 – The Automatic Photo Booth Hands On Lab

In this short, hands-on, coding lab you will use Windows 10 and the Universal Windows Platform to build a photo booth with a few automatic features using more natural ways of interacting with a device. You’ll build on a PC but the code that you write could work on other Windows devices like a phone, XBOX one or Raspberry PI 2.

Here’s a quick tour of the features that you are going to build in the next 45-50 minutes;

1. A camera that turns itself on/off in response to seeing a human face.
2. A speech recognition system that will respond to commands;
   1. “CHEESE!” to take a photo
   2. “CAPTION” to recognize hand-written ink, turn it into text and display it on the photo.
   3. “SAVE” to save the photo to a file.
   4. “RESET” to start over.

The application will support digital ink using pen (ideally) but also touch (pretty good) and mouse which can be a bit tricky to write with but most PCs have a mouse so we’ll support it in case your device does not have pen or touch.

## What You Need To Be Successful – Hardware and Software

In order to get the most from this lab, you will need;

1. A PC or Mac running Windows 10.
2. A working webcam and microphone on your PC in order to capture audio, video and photos.
3. The Visual Studio 2015 Update 1 development environment including the **“Universal Windows App Development Tools”.**
4. An internet connection to do a small download.

If you are in the lab environment and are missing one or more of these pre-requisites, please ask one of the staff who will do their best to assist you.

## What You Need - Skills

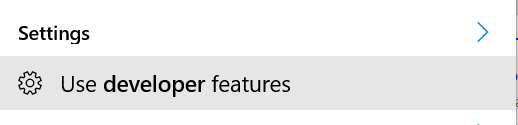
This lab document walks you through building the photo booth using an interface technology called XAML and the C# programming language but you do not need a lot of experience in these technologies or in Visual Studio as the lab will walk you through the process in a step-by-step manner.

You can always stop and ask one of the staff who’ll be more than happy to help.

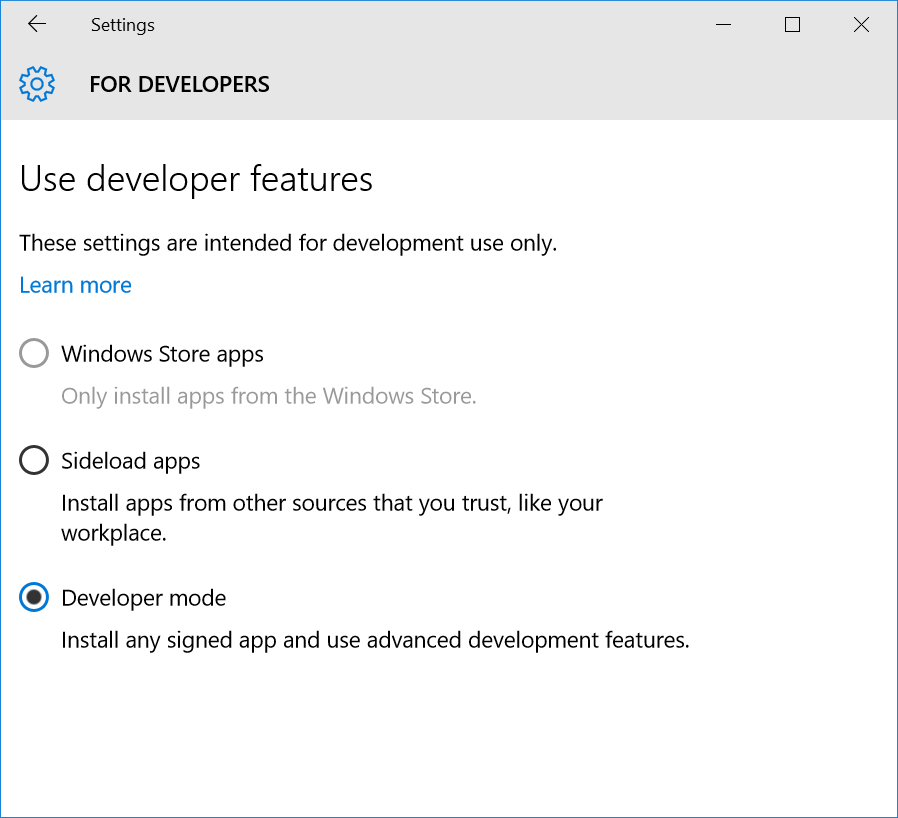
NB: The focus of this lab is to write a *small* amount of code – we will not always write the *best* code to achieve a scenario and short-cuts will be taken around error handling, return types and dealing with asynchronous code.

## Step 1 – Developer Mode

Before starting the lab document, ensure that your Windows 10 machine is in **developer mode**. You can do this by hitting the Windows Key and then beginning to type the word **“Developer”** and the search box should bring up the setting;

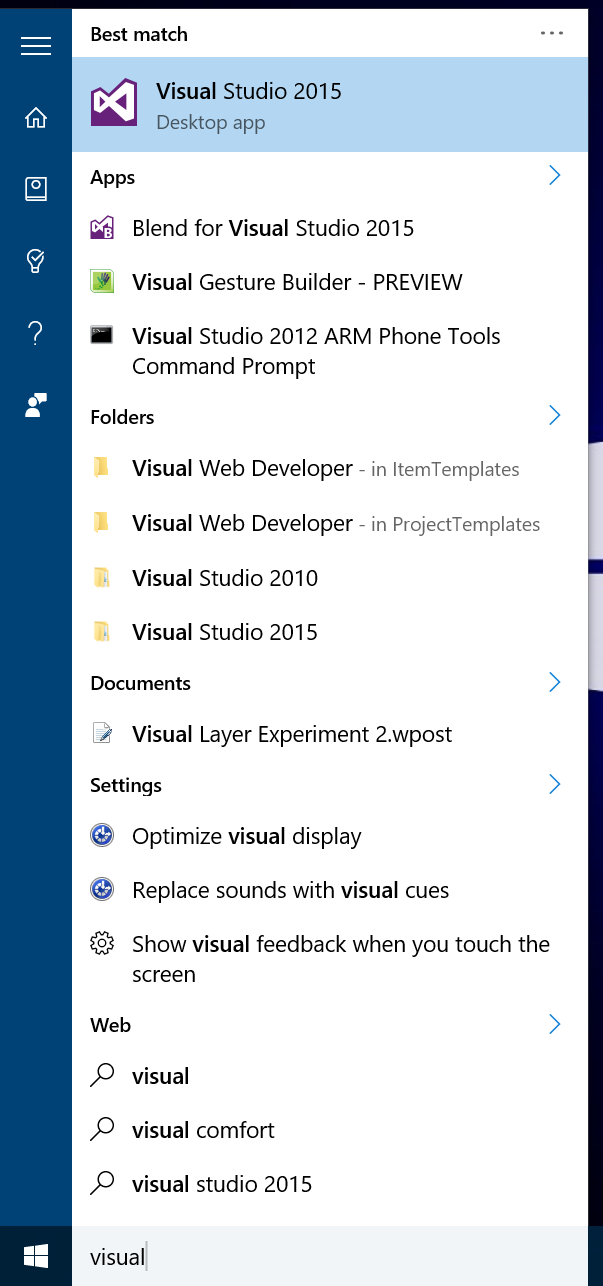


Select this setting and use the control panel to switch on developer mode as below;

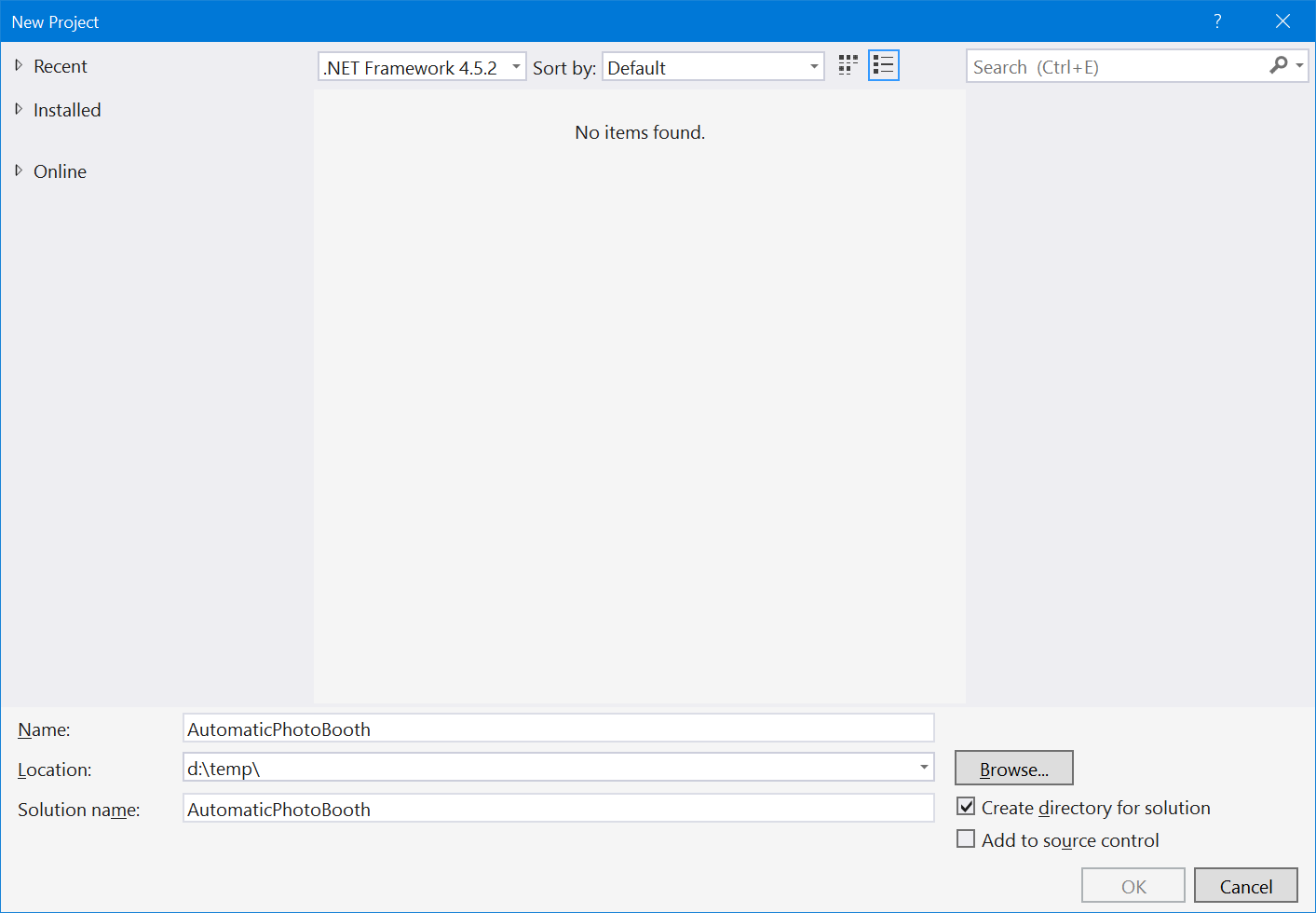


## Step 2 – Making a Blank App Project for Windows 10

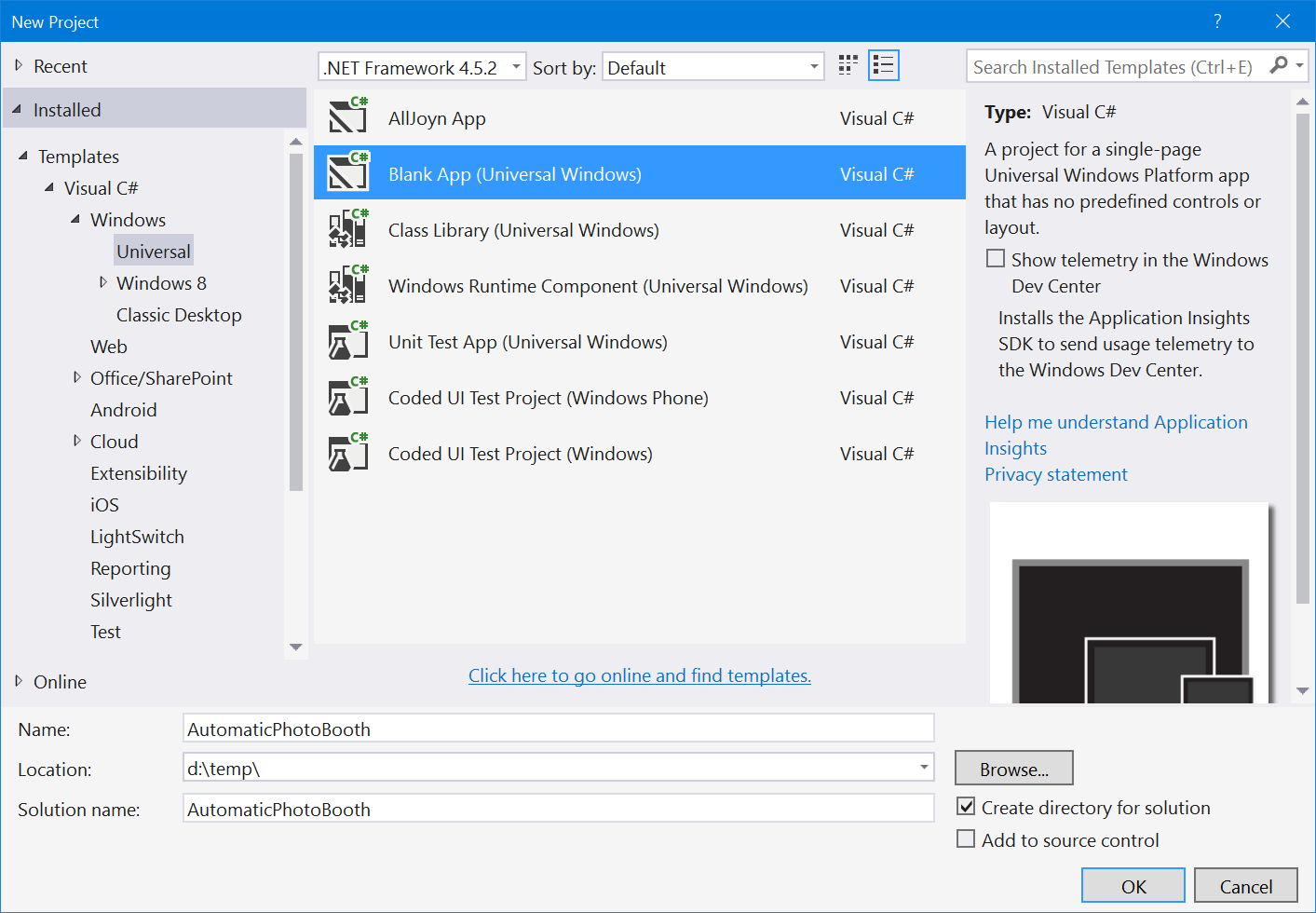
1. Run Visual Studio in Windows 10. You can do this by simply hitting the Windows Key and then beginning to type the word **“Visual”** and the search box should bring up Visual Studio for you to run if you have it installed as shown in the screenshot below;



1. Within Visual Studio, use the **File** menu and select the **New** sub menu and then choose **Project** in order to raise the new project dialog as below;



1. This dialog is driven by the template selection in the tree on the left hand side and so expand out the tree from the **Installed** node such that you expand all the way to **Installed->Templates->Visual C#->Windows->Universal** and that the dialog displays a set of templates as shown below;



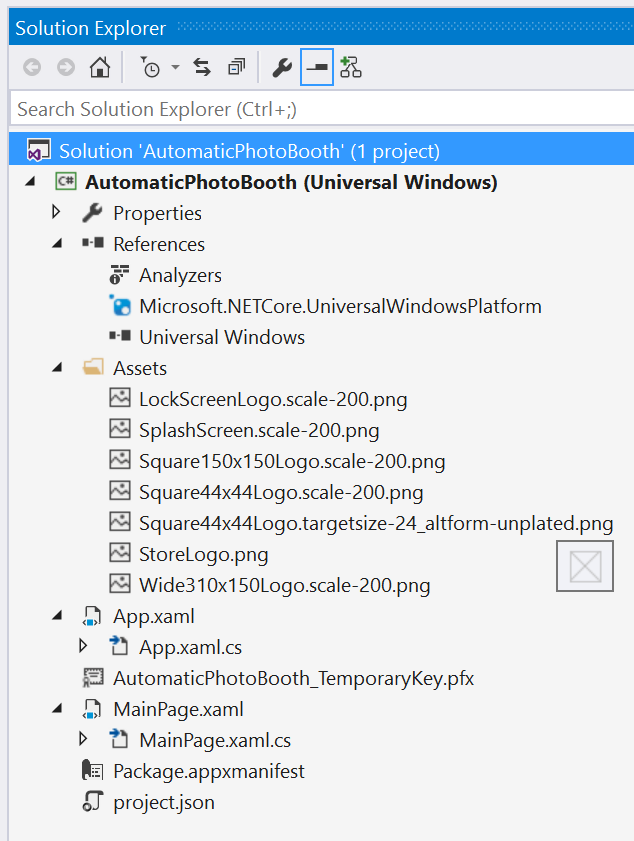
1. Now, choose the **Blank App** **(Universal Windows)** template and then complete the details at the bottom of the dialog in order to provide;
   1. **Name: AutomaticPhotoBooth**
   2. **Location:** <pick a suitable location on your PC like the desktop or a temporary folder>
   3. **Solution name:** Let Visual Studio complete this for you.
2. Hit OK to make the blank project, it will take a few seconds to complete.

## Step 3 – A Tour of Your Blank Project

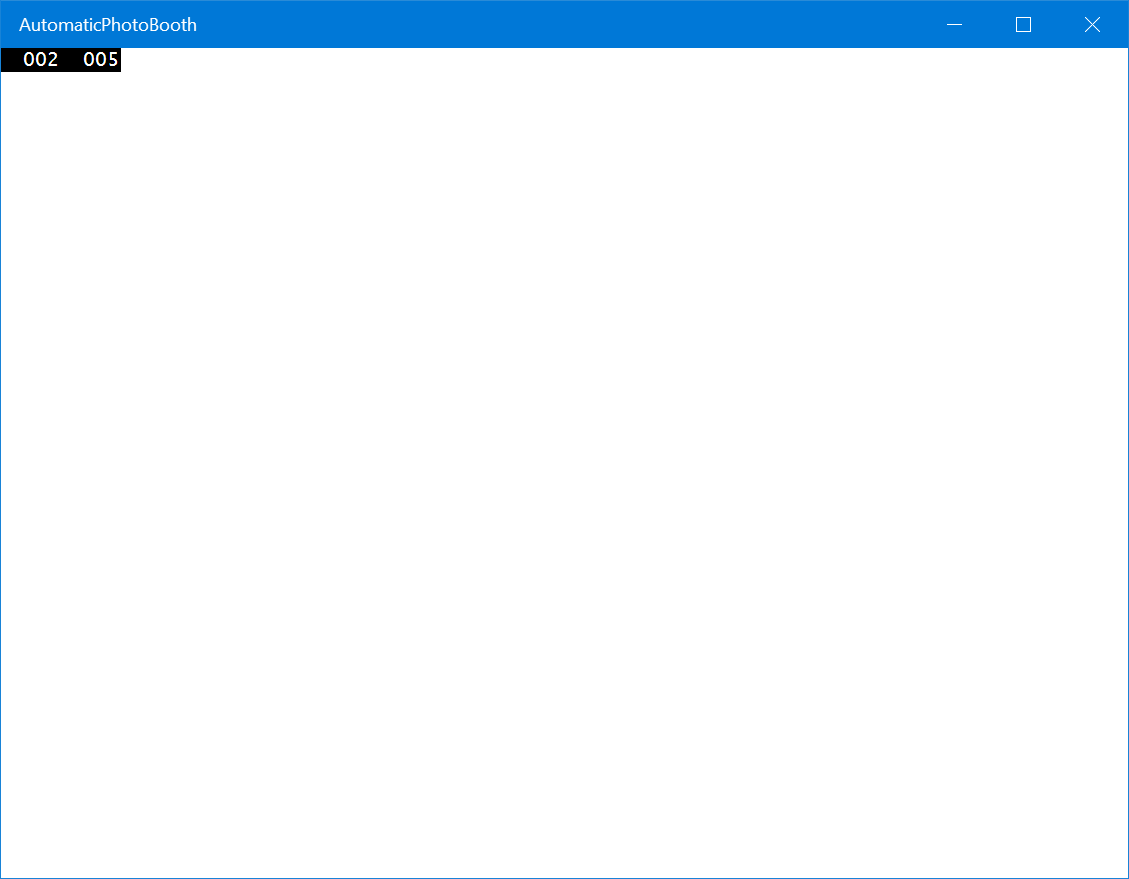
Having created your blank project, let’s take a quick tour around it. If you want to follow along, you will need to open Visual Studio’s **Solution Explorer** window which you can find on the;

* **View** menu, option **Solution Explorer**.

The **Solution Explorer** provides a hierarchical view of your project’s contents and looks something like the screenshot below when expanded on a blank project similar to the one that you have created;



1. Within this tree view you can see the major items that make up your project, described briefly below;
   1. The **App.xaml** and **App.xaml.cs** files contain XML and code respectively which define the **App** class for your application. You will find little in the **.xaml** file but quite a bit of code in the **.xaml.cs** file with most of that code relating to getting the application up and running in the first instance and making its window.
   2. The **MainPage.xaml** and **MainPage.xaml.cs** files contain XML and code respectively which define the main UI for your application. If you open the **.xaml** file then you will be presented with a graphical design surface for designing it where you will find a largely blank UI surface. The code file contains almost no code at this point.
   3. The **Package.appxmanifest** file is a description of your application in terms of its names, icons and its interactions with the system controlling things like security settings.
   4. The **Assets** folder contains some default images that are referenced from the **.appxmanifest** file.
   5. The **References** item has referenced to libraries that your application can call into – in this case, the key items are that you can use some **.NET** libraries and you can use the **Universal Windows** libraries.
2. Before you being coding, ensure that you can run the blank project that you have created on your machine.
   1. Use the **Debug** menu, option **Start Debugging** and Visual Studio will;
      1. Compile the code.
      2. Package the application.
      3. Install it onto the local machine.
      4. Run it up and attach the debugger.
3. Once the application is running, you should see a blank window with some performance counters;

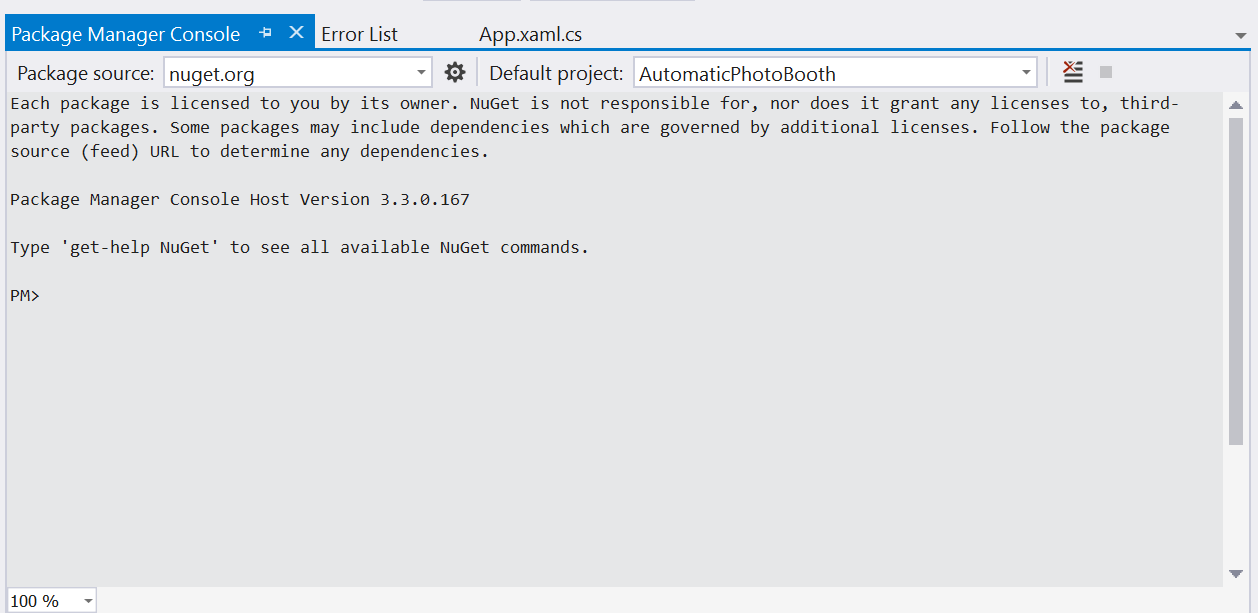


1. Stop the application either by closing it or using the **Debug** menu, **Stop Debugging** option.

## Step 4 – Capturing Video Frames from the Webcam

In order to capture and display video, you are going to make use of a control that has been pre-written to help you out.

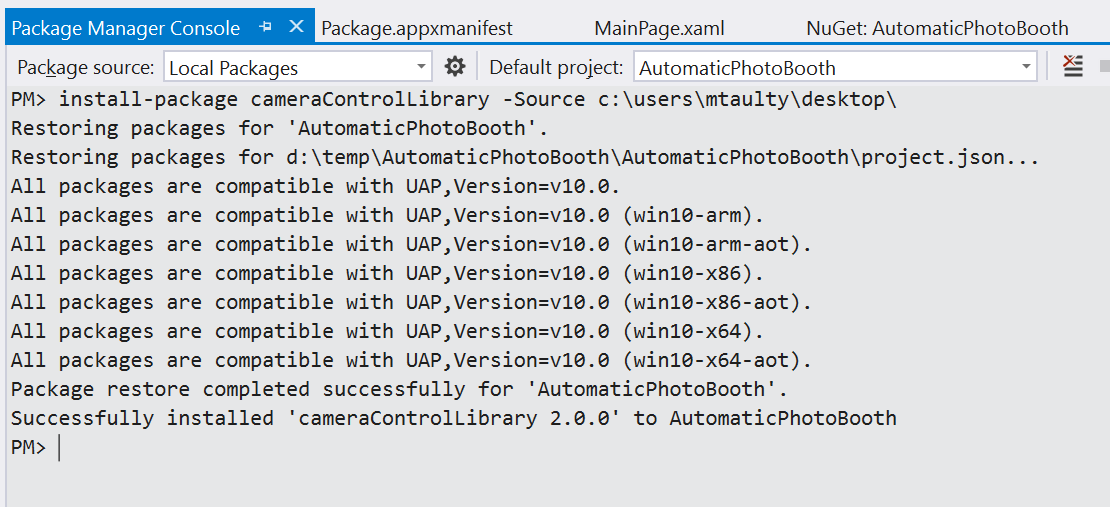
1. Install the control by using the **Nuget** package management solution as below;
   1. Open up a browser and download the file from the URL below;
      * [**http://mtaultyfiles.azurewebsites.net/downloads/booth.zip**](http://mtaultyfiles.azurewebsites.net/downloads/booth.zip)
   2. Unzip the file to your desktop.
   3. Raise the package management console by using the **Tools** menu, **Nuget Package Manager** option, **Package Manager Console** option. You should see a prompt as below;



* 1. Install the package from your desktop using this command line substituting your user name for the one given here (mtaulty);

install-package cameraControlLibrary -Source c:\users\mtaulty\desktop\

* 1. You should see output similar to below;



* 1. Now, add an instance of the control that is in the library to your main UI. Open up your **MainPage.xaml** file. Within there, you will find a blank **Grid** element;

<Grid

Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

</Grid>

* 1. Add a single element to this **Grid** to display the camera output;

<Grid

Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<!-- Newly added line -->

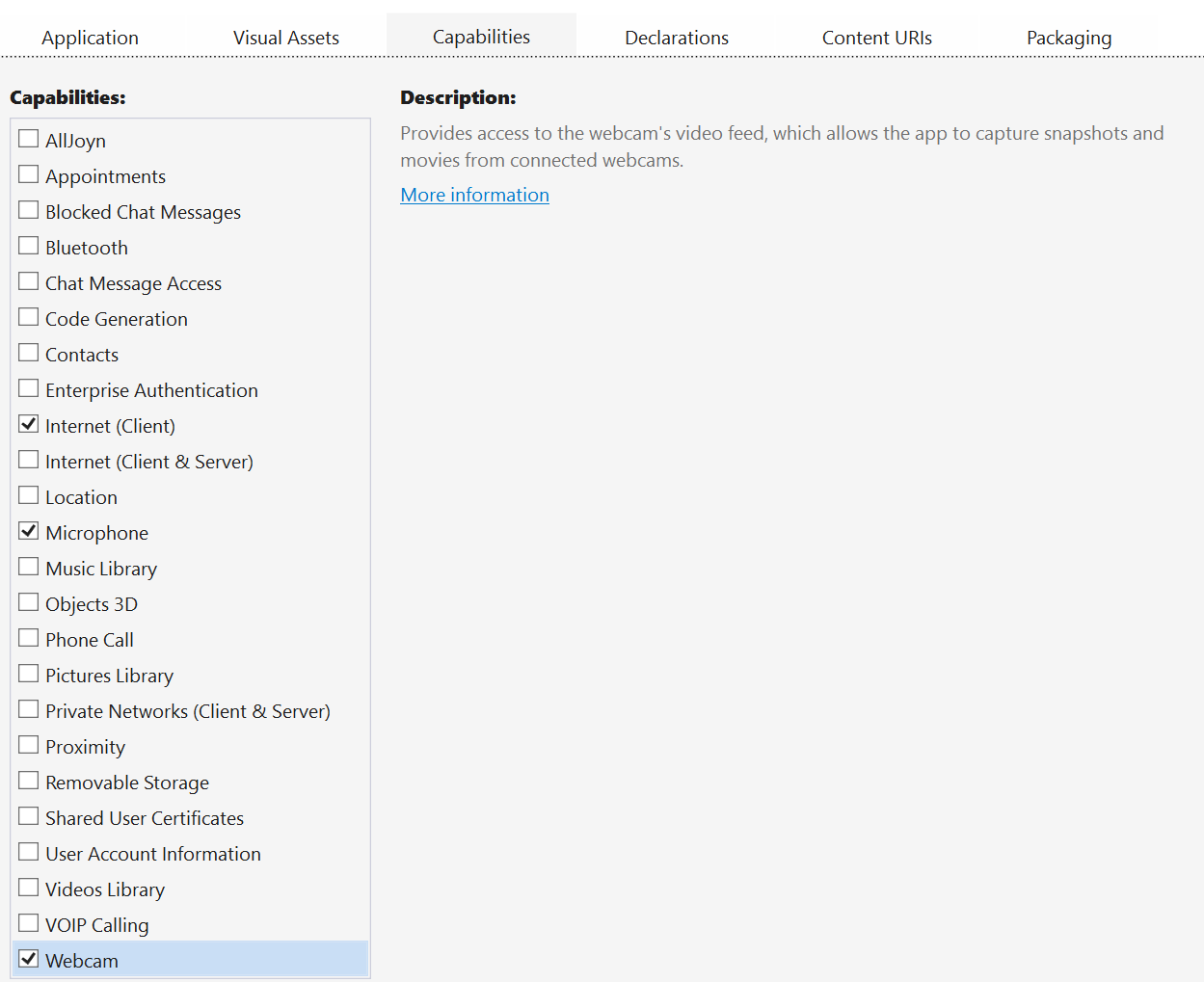
<control:CameraDisplay

x:Name="myCameraControl"

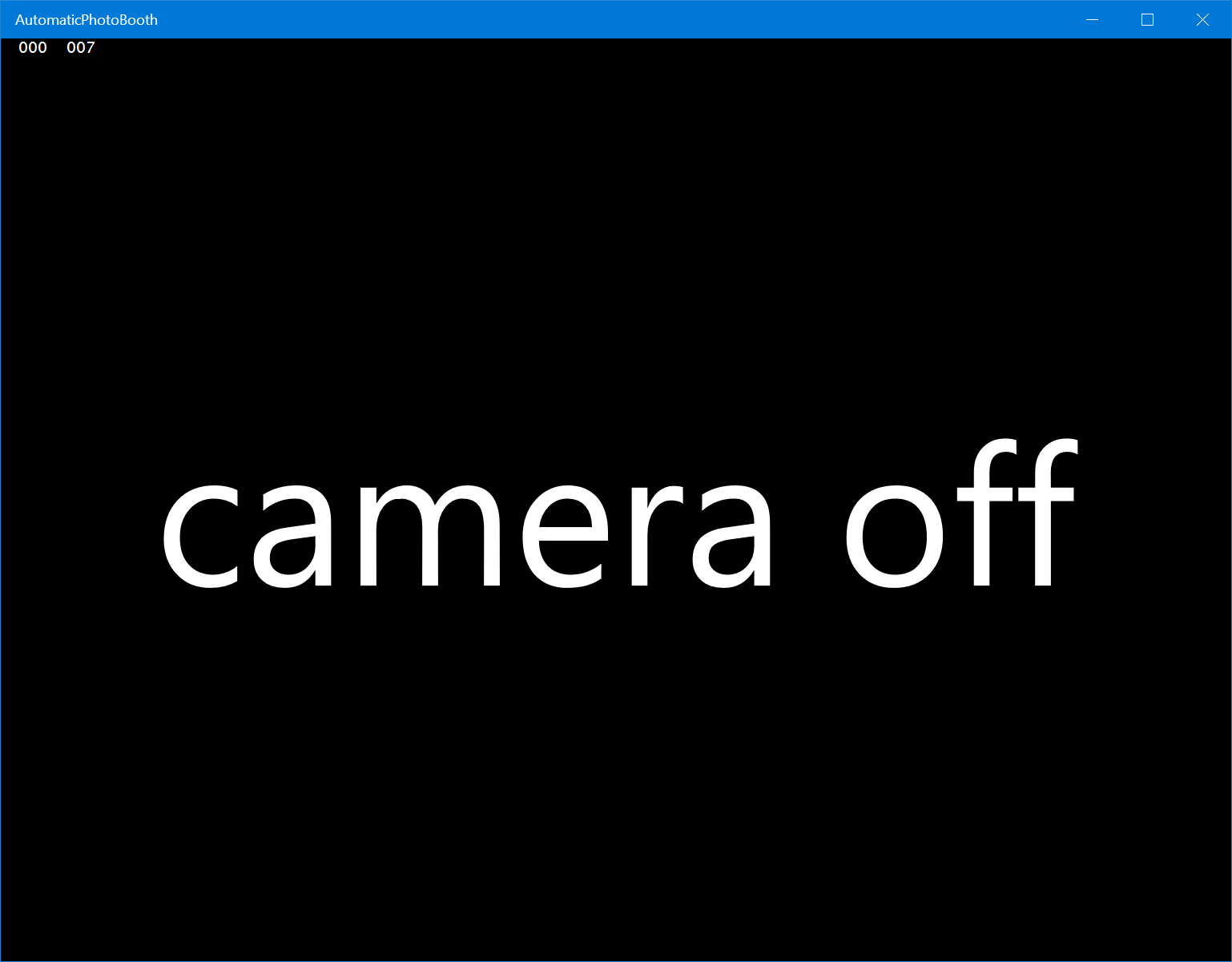
xmlns:control="using:CameraControlLibrary" />

</Grid>

* 1. The control that you have added needs access to the **webcam** and the **microphone** on the PC and so you need to make sure that your application has that access to those features;
     + Open up the **package.appxmanifest** file.
     + Go to the **Capabilities** tab and ensure that both **Microphone** and **Webcam** are ticked as below;



* 1. Save all your files and run the application using the **Debug** menu, **Start Debugging** option as before. You should see a window as below;



## Step 5 – A Face-Acitvated Camera

It’s time to now switch on the camera’s display when it comes across a human face. In order to do this, you are going to write code which processes every video frame from the webcam;

1. Open up the code file **MainPage.xaml.cs**. Within there, you should have a blank class;

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

}

}

1. Add a function which is going to process each frame of video from the camera – the new lines of code are highlighted below, notice that the call the **SetFaceProcessor** method on the **myCameraControl** control to register your function with the control which will then call your function each time it gets a video frame from the webcam.

using System.Threading.Tasks;

using Windows.Graphics.Imaging;

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

this.myCameraControl.SetFaceProcessor(this.ProcessVideoFrame);

}

async Task ProcessVideoFrame(SoftwareBitmap bitmap)

{

}

}

1. Now, you will implement the **ProcessVideoFrame** function by processing each video frame from the camera and deciding whether to show the frame or not based on whether you can detect a face in the frame. First, declare a **FaceDetector** member variable of the class as below;

using System.Threading.Tasks;

using Windows.Graphics.Imaging;

using Windows.Media.FaceAnalysis;

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

this.myCameraControl.SetFaceProcessor(this.ProcessVideoFrame);

}

async Task ProcessVideoFrame(SoftwareBitmap bitmap)

{

if (this.faceDetector == null)

{

this.faceDetector = await FaceDetector.CreateAsync();

}

}

FaceDetector faceDetector;

}

1. Next, implement the face detection code which is not nearly as hard as you might expect given the functionality;

async Task ProcessVideoFrame(SoftwareBitmap bitmap)

{

if (this.faceDetector == null)

{

this.faceDetector = await FaceDetector.CreateAsync();

}

var results = await this.faceDetector.DetectFacesAsync(bitmap);

var showVideo = results?.Count > 0;

this.myCameraControl.ShowCamera(showVideo);

}

1. Now, run your application with the **Debug** menu, **Start Debugging** option and see if you can switch the camera on/off with your face.

## Step 6 – Highlight the Face in the Frame

The camera display control that you are using has a method which will draw a box for you which you can use to highlight the face that it is detecting.

1. Add an extra line of code to include this highlight in your **ProcessVideoFrame** function;
2. Run your application with the **Debug** menu, **Start Debugging** option and see if your code is correctly tracking your face around the screen with a **white, dashed rectangle.**

async Task ProcessVideoFrame(SoftwareBitmap bitmap)

{

if (this.faceDetector == null)

{

this.faceDetector = await FaceDetector.CreateAsync();

}

var results = await this.faceDetector.DetectFacesAsync(bitmap);

var showVideo = results?.Count > 0;

this.myCameraControl.ShowCamera(showVideo);

if (showVideo)

{

this.myCameraControl.HighlightFace(results[0].FaceBox);

}

}

## Step 7 – Adding Voice Control

The camera display control that you are using has a method which can take a static photo for you which you will activate using speech.

1. Add a function called **ListenForCommands** to your class like the one listed below and ensure to call it from the constructor for your class – in this code you will create a **SpeechRecognizer** and then constrain it to listen for the commands that we are interested in. From there, you will cause it to loop in an asynchronous manner, delivering commands as they are “heard”.

using Windows.Media.SpeechRecognition;

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

this.myCameraControl.SetFaceProcessor(this.ProcessVideoFrame);

this.ListenForCommands();

}

async Task ListenForCommands()

{

var recognizer = new SpeechRecognizer();

recognizer.Constraints.Add(

new SpeechRecognitionListConstraint(

new string[] { "cheese", "reset", "save", "caption" }));

await recognizer.CompileConstraintsAsync();

while (true)

{

var results = await recognizer.RecognizeAsync();

if (results?.Confidence == SpeechRecognitionConfidence.High)

{

// TODO: do something with the recognition results.

}

}

}

1. Now, add some code to do something with the commands that are heard. Firstly, use the **“cheese”** command to cause the camera control to take a snapshot. The additional code is highlighted below and replaces the “TODO” comment in the previous code;

if (results?.Confidence == SpeechRecognitionConfidence.High)

{

switch (results.Text)

{

case "cheese":

this.myCameraControl.Snap();

break;

default:

break;

}

}

1. Run your application with the **Debug** menu, **Start Debugging** option. Once running, enable the camera with your face and then say **“cheese”** to cause the control to display the snapped photo.
2. NB – if you are having problems with getting speech recognized then you might need to set up your microphone for speech. Follow these steps;
   1. Press Windows key and type “**microphone**” and then select the “**Microphone Setup**” option.
   2. In the “**Time & Language**” dialog raised, make sure that the highlighted section is “**Speech**” and scroll down to “**Microphone”** and click the “**Get Started”** option to set up your microphone for speech recognition.

## Step 8 – Adding an Ink Overlay

Having got an initial voice command working, you’ll now add digital ink into the application such that a legend or title can be added to your snapped photo using ink.

1. Firstly, open your **MainPage.xaml** user interface definition and add a new **InkCanvas** control into the user interface as below;

<Grid

Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<control:CameraDisplay

x:Name="myCameraControl"

xmlns:control="using:CameraControlLibrary" />

<!-- Newly added line -->

<InkCanvas

x:Name="myInkCanvas" />

</Grid>

1. Next, open up your **MainPage.xaml.cs** file and add some code to get the basics of ink processing into the application. Firstly, alter the constructor such that the new **InkCanvas** is set up to handle pen, mouse and touch (by default, it will only do pen);
2. Now, write a function which recognizes the ink and tells the camera control to display the recognized text as a legend on the photo. This function takes all the ink that is currently captured on screen, turns it into text and then clears the ink from the screen;

using Windows.UI.Input.Inking;

public sealed partial class MainPage : Page

{

async Task RecogniseInk()

{

var recognizer = new InkRecognizerContainer();

var results = await recognizer.RecognizeAsync(

this.myInkCanvas.InkPresenter.StrokeContainer,

InkRecognitionTarget.All);

var recognisedText = results?[0]?.GetTextCandidates()?.FirstOrDefault();

if (!string.IsNullOrEmpty(recognisedText))

{

this.myCameraControl.ShowLegend(recognisedText);

}

this.myInkCanvas.InkPresenter.StrokeContainer.Clear();

}

using System.Threading.Tasks;

using Windows.Graphics.Imaging;

using Windows.Media.FaceAnalysis;

using Windows.Media.SpeechRecognition;

using Windows.UI.Core;

public sealed partial class MainPage : Page

{

public MainPage()

{

this.InitializeComponent();

this.myCameraControl.SetFaceProcessor(this.ProcessVideoFrame);

this.ListenForCommands();

this.myInkCanvas.InkPresenter.InputDeviceTypes =

CoreInputDeviceTypes.Pen |

CoreInputDeviceTypes.Touch |

CoreInputDeviceTypes.Mouse;

}

1. Finally, call this function via the speech command **“caption”** by adding that into the switch statement that you added in the last section of the lab when you first set up speech recognition;

async Task ListenForCommands()

{

var recognizer = new SpeechRecognizer();

recognizer.Constraints.Add(

new SpeechRecognitionListConstraint(

new string[] { "cheese", "reset", "save", "caption" }));

await recognizer.CompileConstraintsAsync();

while (true)

{

var results = await recognizer.RecognizeAsync();

if (results?.Confidence == SpeechRecognitionConfidence.High)

{

switch (results.Text)

{

case "cheese":

this.myCameraControl.Snap();

break;

case "caption":

await this.RecogniseInk();

break;

default:

break;

}

}

}

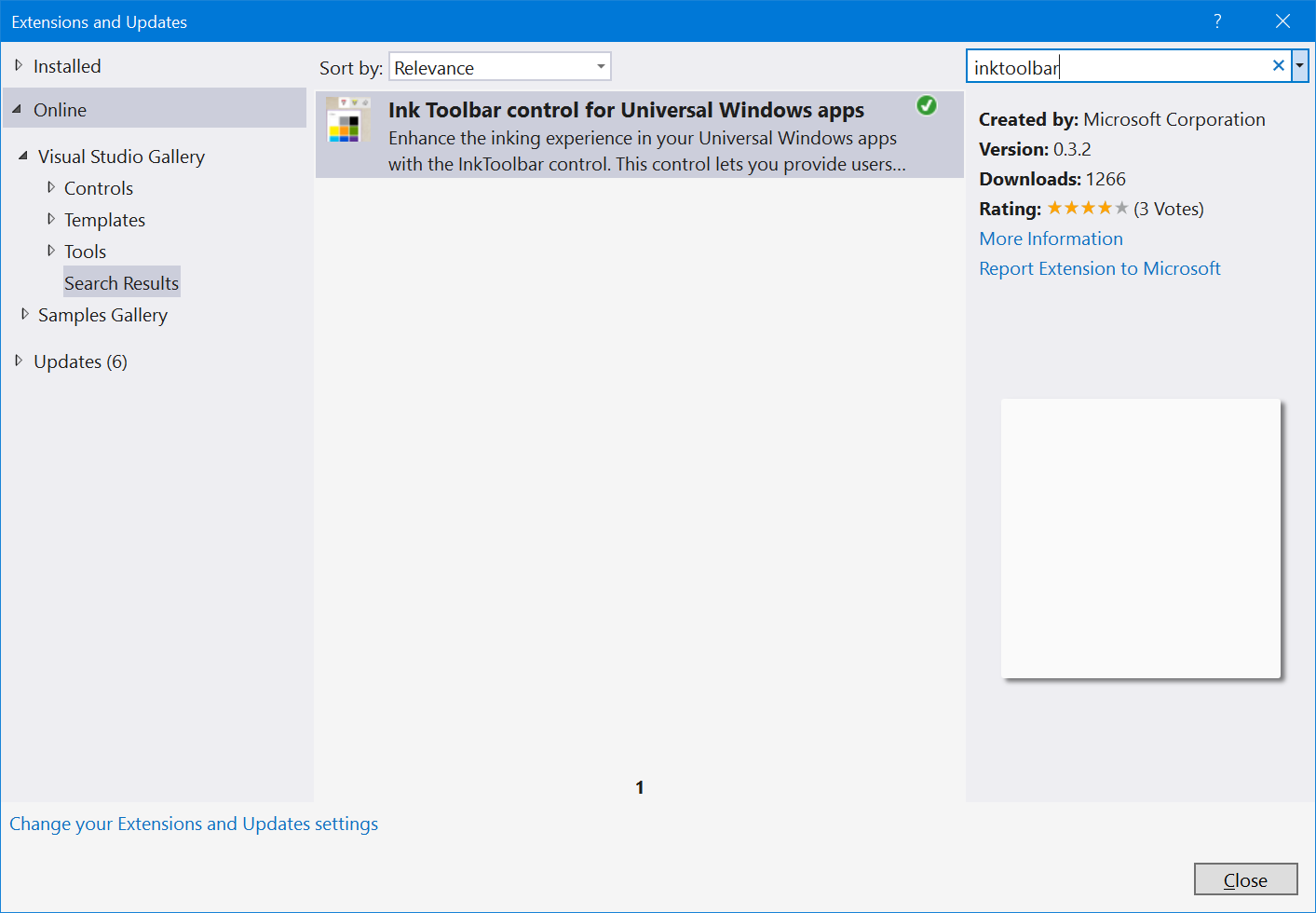
}

1. Run your application with the **Debug** menu, **Start Debugging** option. Once the application is running, try it out with the follow sequence of actions;
   1. Activate the camera with your face.
   2. Say “cheese” to snap a photo.
   3. Draw on the photo with your pen, finger or mouse to write some text like “smile”.
   4. Say “caption” to display the caption on the photo.
   5. Note that the ink will be displaying in black by default so may be a little tricky to see, you will fix this in the following (optional) step.

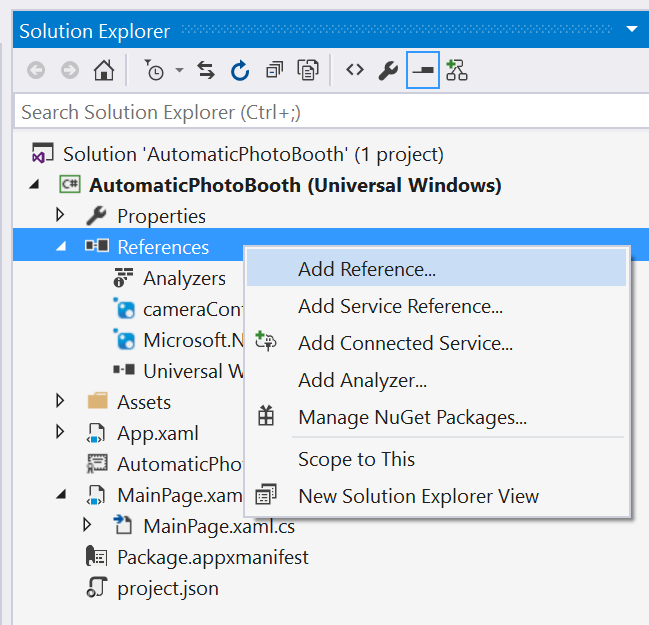
## Step 9 (Optional) – Control and Erase Your Ink

In the previous step, you added the basics of ink into your photo booth but it is easy to take control of your ink and get formatting and draw/erase for free. You’ll add that functionality in this step of the lab using a control that you can download from the web called the **InkToolbarControl**.

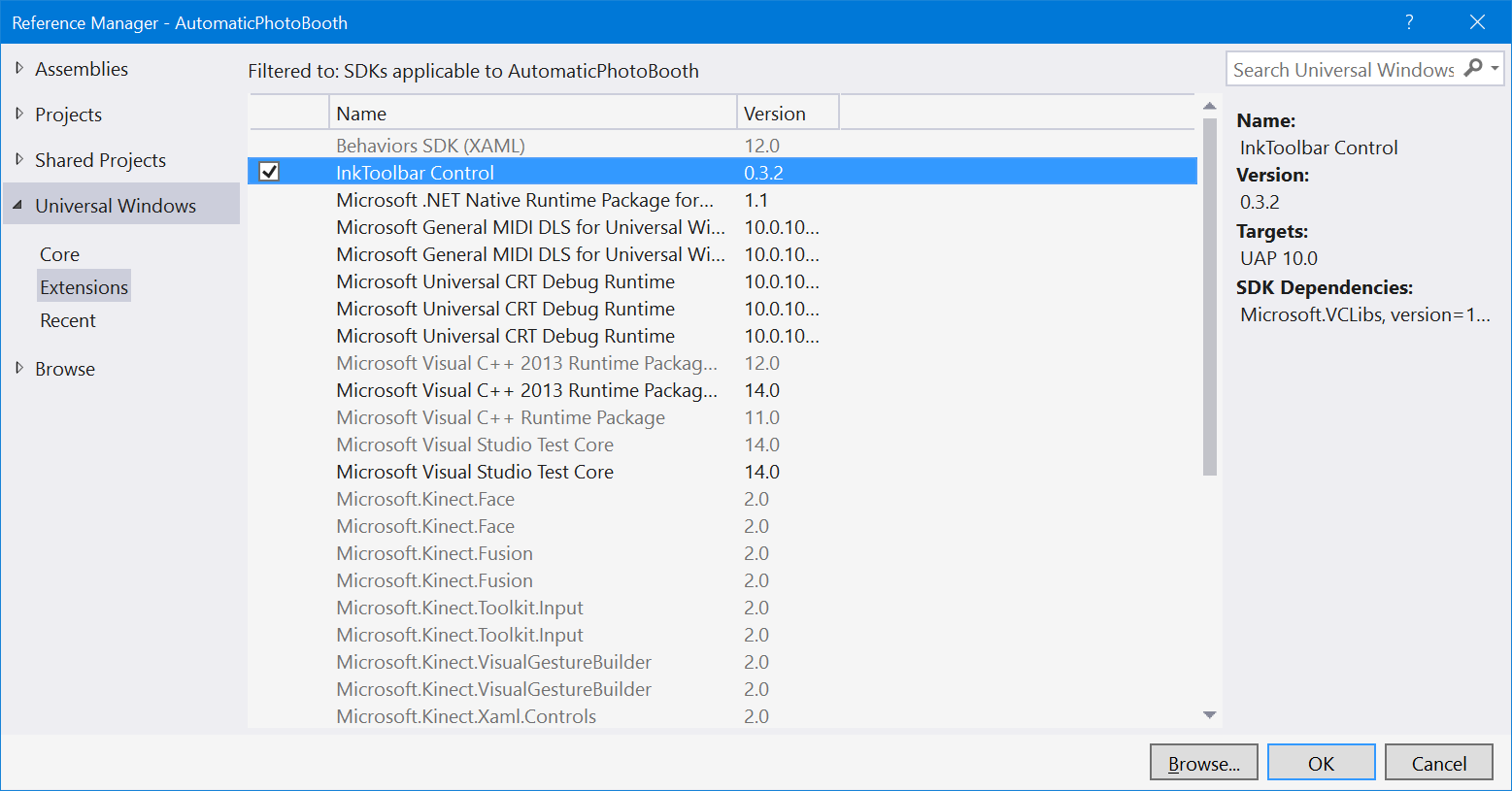
1. Use the **Tools** menu, **Extensions and Updates** option to raise the **“Extensions and Updates**” dialog in Visual Studio;



1. Use the tree on the left hand side to select the **Online**->**Visual Studio Gallery** area and then use the search box in the top right to search for **inktoolbar**.
2. When you find the control, install it into your Visual Studio (you may need to restart Visual Studio and re-open your project to complete these step).
3. Next, you will need to add this control into your **project** by going back to the **Solution Explorer** (from the **View** menu, option **Solution Explorer**) and then using the right mouse menu on your project’s **references** in order to choose the **Add Reference** option;



1. In the **“Reference Manager”** dialog that is raised, use the tree on the left to select the **Universal Windows->Extensions** node and then make sure that you tick the **InkToolbar Control** in the list in the middle before clicking OK;



1. Now, with the Ink Toolbar Control present in your project, revisit your UI by opening your **MainPage.xaml** file and declare an instance of this control as per the XAML below. Note that the property **TargetInkCanvas** here is used to connect the **InkToolbar** to the **InkCanvas** control.

<Grid

Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<control:CameraDisplay

x:Name="myCameraControl"

xmlns:control="using:CameraControlLibrary" />

<InkCanvas

x:Name="myInkCanvas" />

<!-- Newly added line -->

<ext:InkToolbar

TargetInkCanvas="{x:Bind myInkCanvas}"

xmlns:ext="using:Microsoft.Labs.InkToolbarControl"

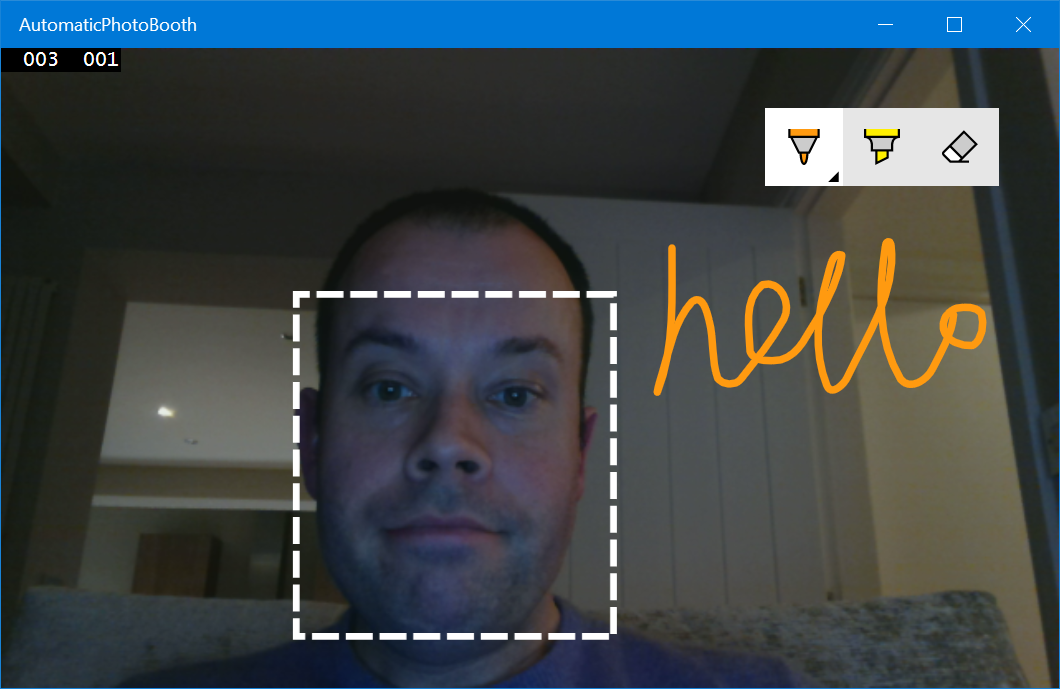
VerticalAlignment="Top"

HorizontalAlignment="Right"

Margin="40"/>

</Grid>

1. Run your application with the **Debug** menu, **Start Debugging** option. Once the application is running, try it out with the follow sequence of actions;
   1. Activate the camera with your face.
   2. Say “cheese” to snap a photo.
   3. Now, draw on the photo as in the previous section of the lab but you should have controls which will change the colour, width of the ink and also which will allow you to erase the ink.



## Step 10 – Adding Save and Reset Commands

The last step here is an easy one – you will add two more commands to the dictionary understood by the speech recognizer such that it is possible to save the photo along with its legend and then you can reset the application back to its original state.

1. Revisit the function that you wrote called **ListenForCommands** and add in the code to handle the remaining two commands – “save” and “reset”. The code below contains the highlighted snipped to add along with some context as to where it belongs;

var results = await recognizer.RecognizeAsync();

if (results?.Confidence == SpeechRecognitionConfidence.High)

{

switch (results.Text)

{

case "cheese":

this.myCameraControl.Snap();

break;

case "caption":

this.RecogniseInk();

break;

case "save":

await this.myCameraControl.SaveAsync();

break;

case "reset":

this.myCameraControl.ResetVisuals();

this.myInkCanvas.InkPresenter.StrokeContainer.Clear();

break;

default:

break;

}

}

1. Test your code – run the app as previously with the **Debug** menu, **Start Debugging** option and then work through the sequence of face recognition, taking a photo with the “cheese” command before adding an ink based caption with the “caption” command and, finally, saving the photo to a file with the “save” command. You can then use “reset” to start again.

Congratulations on reaching the end of the lab – we hope that you saw that it was a long way from rocket-science to add voice, ink and face detection into an app with just a few lines of code.

Go to <http://dev.windows.com> to learn more.