Activate Azure: Intelligent Apps

Scenario: Alpine Ski House Happiness Meter

POC Instructions

Instructor Edition

**Conditions and Terms of Use**

**Microsoft Confidential**

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

**Copyright and Trademarks**

© 2017 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

DirectX, Hyper-V, Internet Explorer, Microsoft, Outlook, OneDrive, SQL Server, Windows, Microsoft Azure, Windows PowerShell, Windows Server, Windows Vista, and Zune are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners

.

Contents

[PBI 1: Using Face REST API 6](#_Toc520913675)

[Introduction 6](#_Toc520913676)

[Objectives 6](#_Toc520913677)

[Prerequisites 6](#_Toc520913678)

[Scenario 6](#_Toc520913679)

[Task 1: Create Face API Resource 6](#_Toc520913680)

[Task 2: Get Face API Keys 9](#_Toc520913681)

[Task 3: Working with Image Stream 10](#_Toc520913682)

[Task 4: Detecting Emotion in Image 10](#_Toc520913683)

[Task 5: Face API Result 11](#_Toc520913684)

[Task 6: Familiarize with Project Oxford libraries 13](#_Toc520913685)

[PBI 2: Face API as A Service Library 14](#_Toc520913686)

[Introduction 14](#_Toc520913687)

[Objectives 14](#_Toc520913688)

[Prerequisites 14](#_Toc520913689)

[Scenario 14](#_Toc520913690)

[Task 1: Add missing Packages 14](#_Toc520913691)

[Task 2: Create an EmotionData class 15](#_Toc520913692)

[Task 3: Edit EmotionServiceHelper class 15](#_Toc520913693)

[Task 4: Edit ImageAnalyzer class 16](#_Toc520913694)

[Task 5: Build your project 17](#_Toc520913695)

[PBI 3: Setting up IntelligentKioskSample 18](#_Toc520913696)

[Introduction 18](#_Toc520913697)

[Objectives 18](#_Toc520913698)

[Prerequisites 18](#_Toc520913699)

[Scenario 18](#_Toc520913700)

[Task 1: Add missing Packages 18](#_Toc520913701)

[Task 2: Adjust application settings 18](#_Toc520913702)

[PBI 4: Detecting Emotion in Image 21](#_Toc520913703)

[Introduction 21](#_Toc520913704)

[Objectives 21](#_Toc520913705)

[Prerequisites 21](#_Toc520913706)

[Scenario 21](#_Toc520913707)

[Task 1: Run your application 21](#_Toc520913708)

[Task 2: Wiring up Emotion Detection 21](#_Toc520913709)

[Task 3: Add emoji Control 22](#_Toc520913710)

# PBI 1: Using Face REST API

#### Introduction

In this PBI, you will explore the Microsoft Cognitive Services Face API in a console application. This is the typical “Hello World” approach to understand what you will need and the development process in detecting emotion.

#### Objectives

After completing this lab, you will be able to:

* Familiarize with the Microsoft Cognitive Services Face API.

#### Prerequisites

Checkout **PBI1-Begin** branch of AlpineSkiHouseHappinessMeter folder from GitHub

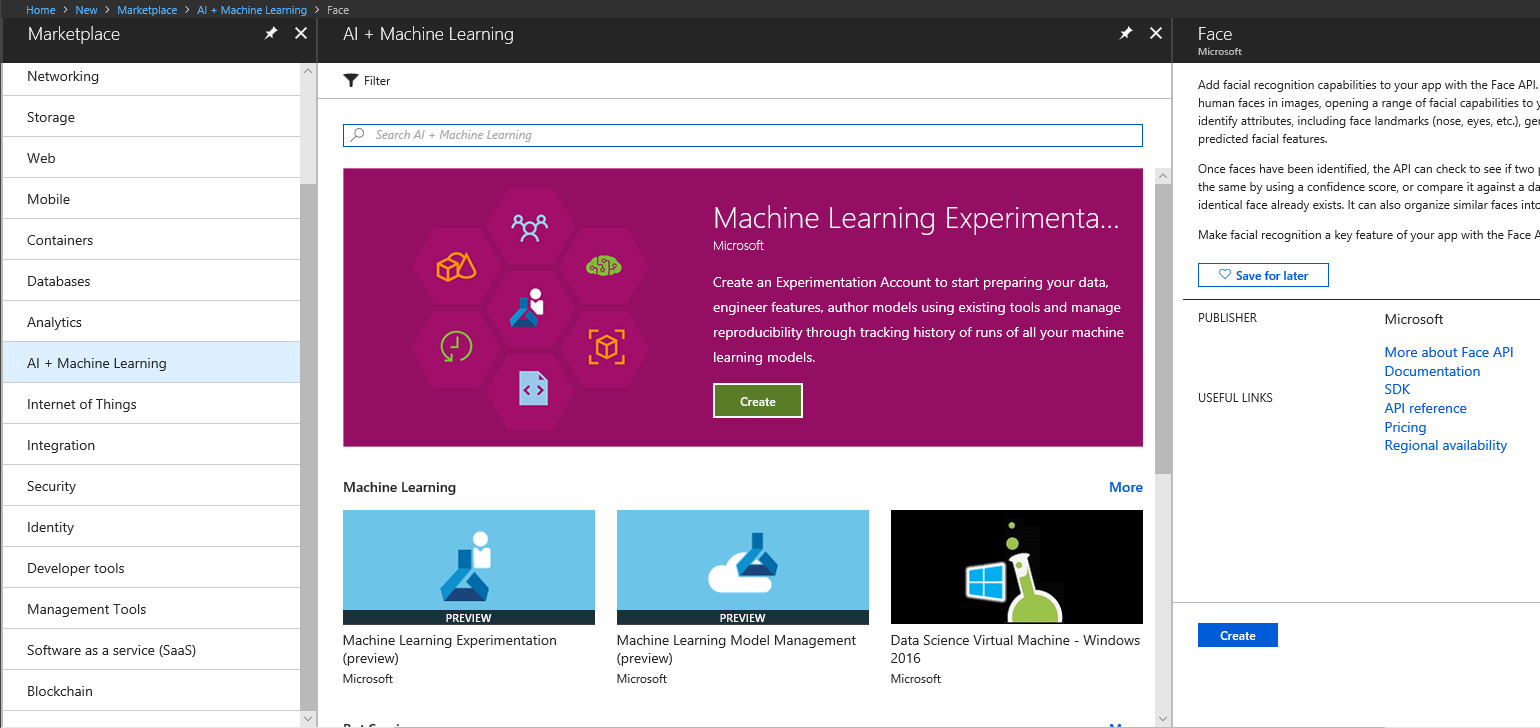
#### Scenario

To start using the services provided by the Microsoft Cognitive Services Face API it will be necessary to create a resource first before you can use it. This PBI will help you to create the required resource to be able to use it in your application. The Face API takes a facial expression in an image as an input and returns the confidence across a set of emotions for each face in the image, as well as bounding box for the face, using the Face API. If a user has already called the Face API, they can submit the face rectangle as an optional input.

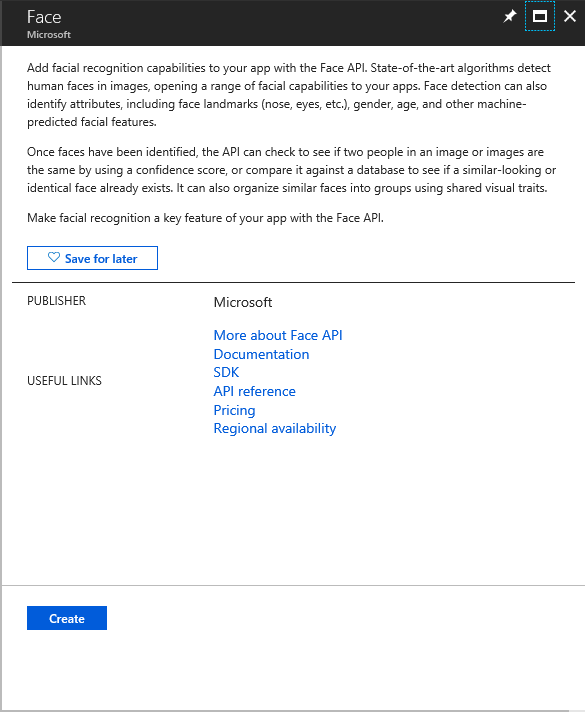
The emotions detected are anger, contempt, disgust, fear, happiness, neutral, sadness, and surprise. These emotions are understood to be cross-culturally and universally communicated with particular facial expressions.

Task 1: Create Face API Resource

1. Sign in into Microsoft Azure Portal by using your account <http://portal.azure.com>
2. You will create a new Microsoft Cognitive Services Face API, locate the button Create a resource in Azure portal and click on it.
3. In the New blade search “Face API” or locate it by browsing “AI + Machine Learning”, “Cognitive Service” category:



1. The Face API landing blade will show you the description of this resource and the legal terms associated, also you will be presented with information links that will help you understand how the API works. Click the Create button.



1. Use the following setting to create your resource:

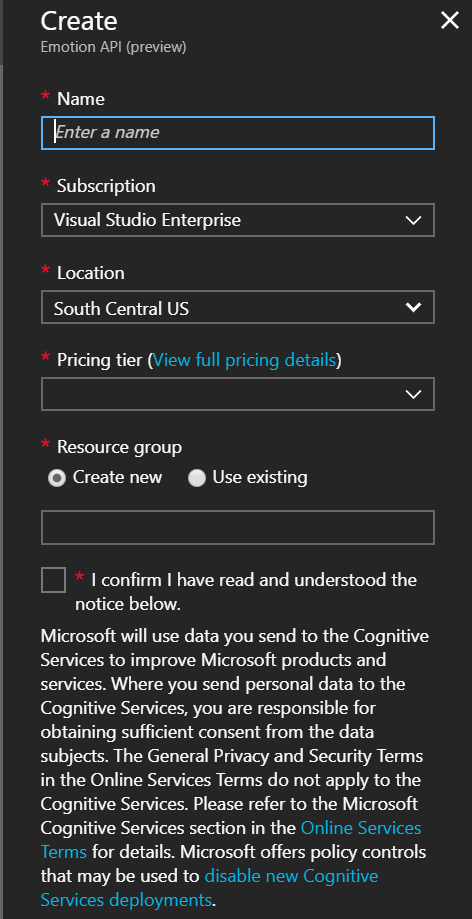
**Name:** AlpineSkiHouseAPI

**Subscription:** <Select your subscription>

**Location:** West US

**Pricing Tier:** S0

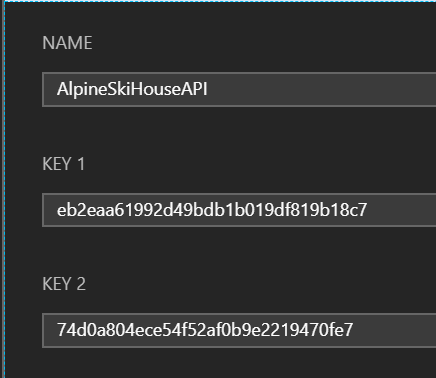
**Resource Group:** Create New (AlpineSkiHouseRG)



Task 2: Get Face API Keys

To communicate with the Face API resource that you just created on the Task 1, you will need an identification that let Cognitive Services know that the request is valid, comes from a trusted application and will have some specific configurations set. This will improve security as you can only access and use this resource if you have a valid key, otherwise your request will be rejected.

1. Once the resource was created locate it on the Azure Portal and open it.
2. Now locate the Keys menu and copy one of the keys to Notepad. You will need it in the later task.



1. This option will always have 2 different keys, if you need to regenerate one of the keys you can start using the second key in the application and no service affectation will take place.

Task 3: Working with Image Stream

1. Locate the *GetImageAsByteArray* method in the *EmotionAPI\_Test* console application, you are going to create the necessary implementation to open an image file by using a path and return an array of bytes that later will be sent to Face API to emotion recognition.
2. Create a new *FileStream* by using the parameter *imageFilePath*. You should open it with an Open mode and a Read file access.
3. Create a new *BinaryReader* by using the *FileStream* created on the step 2.
4. Return the result of the *ReadBytes* method call.

Task 4: Detecting Emotion in Image

1. Locate the *EmotionAPI\_Test* application.
2. Install the *Newtonsoft.Json* NuGet package on this project
3. Inside the *CallEmotionAPI* method declare a new *HttpClient* object.
4. Add a new *DefaultRequestHeaders* item by using the “*Ocp-Apim-Subscription-Key”* key. This key will identify your calls to the API resource, ensure you pass your Api Key as the value. There is a property named ApiKey that will hold your current ApiKey and that you will setup later.

**HttpHeader**

**Key:** Ocp-Apim-Subscription-Key

**Value:** ApiKey

1. Now declare a string variable to store the API URL address.

https://<region>.api.cognitive.microsoft.com/face/v1.0/detect?returnFaceId=true&returnFaceLandmarks=false&returnFaceAttributes=age,gender,emotion

1. Make a call to the *GetImageAsByteArray* passing the image path parameter using the imageFilePath variable and store the results in a new variable called byteData.
2. Paste the following code after the byte array variable.

HttpResponseMessage JsonResponse;

string responseContent;

// Prepare the image and the request to query the Face API

using (ByteArrayContent imageBytes = new ByteArrayContent(byteData))

{

}

1. Inside the using block assign a new MediaType with the value “application/octet-stream” to the *ByteArrayContent* content type header.
   1. Example:

imageBytes.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

1. Invoke the *PostAsync* method of the declared *HttpClient* object by passing the API URL and the image as a byte array using imageBytes variable. Store the result in *JsonResponse* object.
2. Paste the following line

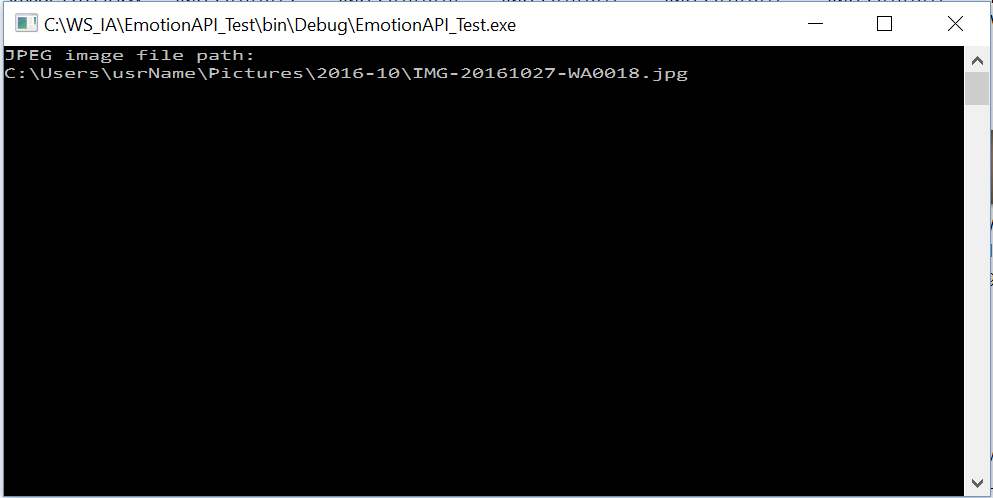
responseContent = JsonResponse.Content.ReadAsStringAsync().Result;

1. Now write some code to show in the console’s window the result of the API call.
2. Now you already have the response, if you want to try your application open the App.config file and replace the apiKey setting with your API key.
3. Now you will use the Newtonsoft.Json library to deserialize this JSON string and display the information on screen, you can paste the following snippet and start working to show the same information of the JSON string using the Newtonsoft library-

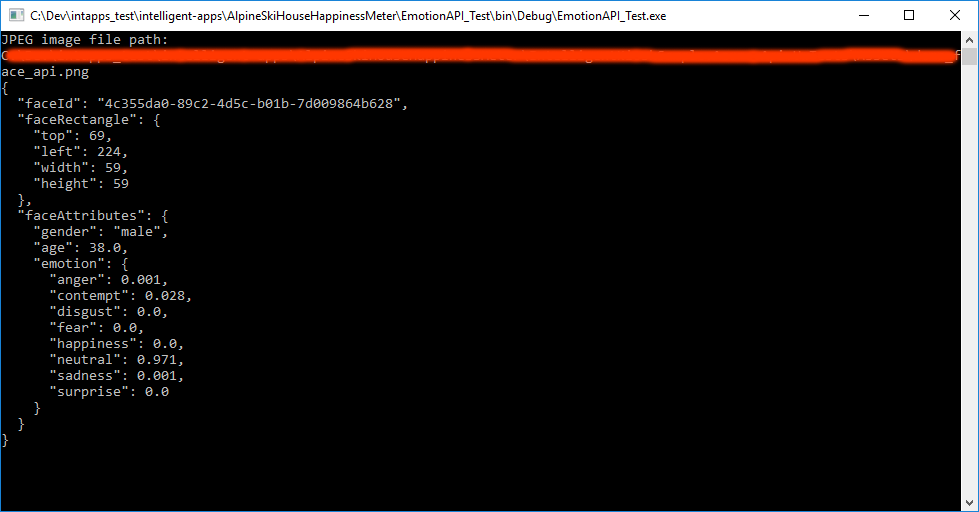
JToken root = JArray.Parse(responseContent).First;

Task 5: Face API Result

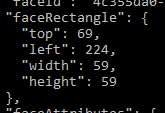
1. Locate the EmotionAPI\_Test application.
2. Open the *App.config* file and replace the apiKey in appSetting value with your API key copied in the Task 2 of this PBI.
3. Run the project (You might need to unload the *IntelligentKioskSample* and *IntelligentKioskSample.Azure.Api* projects to run this project.
4. Provide an image path, this image will need to have a face, so the API can recognize it.



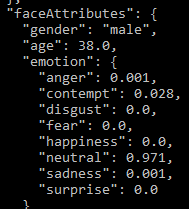
1. Examine the response, this will be the raw JSON string that will have 2 arrays in it,



The first array (faceRectangle) will have the face rectangle detected, these are the coordinates where the Face API detected a face



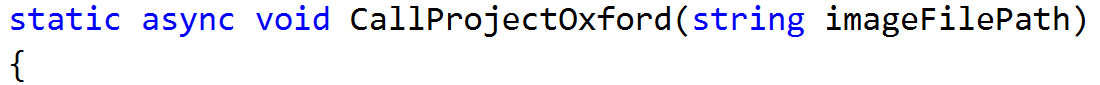
The second array (faceAttributes) will have the detected emotions on the picture



You will need these 2 arrays later to start working with the *Kioks* application. You are going to use the [Microsoft.Azure.CognitiveServices.Vision.Face](https://www.nuget.org/packages/Microsoft.Azure.CognitiveServices.Vision.Face/2.0.0-preview) libraries that will help simplify the processing of requests and JSON objects. That way you can focus on the implementation logic and not in the communication and serialization steps. Consider checking this implementation logic and the JSON string results so you can understand the required process for the following PBIs.

Task 6: Familiarize with Microsoft.Azure.CognitiveServices.Vision.Face NuGet package

1. Now you are going to start working with *Microsoft.Azure.CognitiveServices.Vision.Face* libraries, this way the communication and response deserialization will be encapsulated, and you should not worry about this anymore. Install the library by using NuGet in the *EmotionAPI*\_Test application.
2. Import the namespace *Microsoft.Azure.CognitiveServices.Vision.Face*
3. Declare a new async method named *CallProjectOxford* that receives a string as a parameter.



1. Declare a new *FaceClient* object and pass this code to the constructor:

new ApiKeyServiceClientCredentials(ApiKey),

new System.Net.Http.DelegatingHandler[] { }

1. Now set the clients base uri of your’s Face API (eg: https://<region>.api.cognitive.microsoft.com/face/v1.0) using the FaceClient.*BaseUri* property
2. Now create a new *MemoryStream* by using the result of the *GetImageAsByteArray* method.
3. Now call the *FaceClient.Face.DetectWithStreamAsync* method passing the *memoryStream* object as first argument and wait for the results. At this point you already use the service and are storing the result not in a JSON string but in an emotion object that will make it easier to iterate over it. Also use these values for the other parameters:
   * 1. returnFaceId = true
     2. returnFaceLandmarks = false
     3. returnFaceAttributes = faceAttributes
4. Iterate over the result and print in the console the information about the face rectangle and the detected emotions and scores.
5. Finally add the call to this new method (*CallProjectOxford*) in the main method after the *CallEmotionAPI* method and compare the results.
6. Compare the code between both methods and try to find the steps that are no longer required by using the project oxford libraries, as you can see the project Oxford libraries add simplicity and let us focus on the implementation logic. Debug this console application to understand the differences between both methods, you will need to understand this for later PBIs.

# PBI 2: Face API as A Service Library

#### Introduction

In this PBI, you will use *Microsoft.Azure.CognitiveServices.Vision.Face* library in the *IntelligentKioskSample.Azure.Api* Universal Windows class project. This project is meant to be a helper library for abstracting the Face API’s which will be used in the *IntelligentKiokSample* project.

#### Objectives

After completing this lab, you will be able to:

* Understand how to design a service library to use Face API.

#### Prerequisites

You should finalize the PBI1 activities or checkout **PBI2-Begin** branch of *AlpineSkiHouseHappinessMeter* folder from GitHub

#### Scenario

To start using the services provided by the Microsoft Cognitive Services Face API it will be necessary to add some basic references to work with this APIs in an easier way. The *Microsoft.Azure.CognitiveServices.Vision.Face* library will stablish the communication with the API resource by using the API key, this library will make easier the consumption of the API and the interpretation of the results.

Task 1: Add missing Packages

1. Examine the source code of the *IntelligentKioskSample.Azure.Apis* project, if you try to compile it there will be a lot of exception that you need take care of.
2. Use the NuGet Package Manager to lookup for *Microsoft.Azure.CognitiveServices.Vision.Face and* install the latest version of these components.

Install-Package Microsoft.Azure.CognitiveServices.Vision.Face -Version 2.0.0-preview

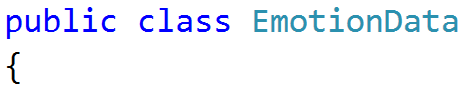
1. Now navigate to *EmotionServiceHelper* class file and add the required using to reference these libraries. Ex:

using Microsoft.Azure.CognitiveServices.Vision.Face;

using Microsoft.Azure.CognitiveServices.Vision.Face.Models;

Task 2: Create an EmotionData class

1. Create a new public class and name it *EmotionData*, this class will be created to store and retrieve the emotion name and score, so you can measure the emotions captured according to your face expressions.



1. As this class will be used to record the emotion name and the emotion score, create 2 public properties with the following values:

**Property 1**

**Name:** EmotionName

**Type:** String

**Property 2**

**Name:** EmotionScore

**Type:** double

Task 3: Edit EmotionServiceHelper class

1. Create a new private static class variable by using the following values, this attribute will be used to access the Microsoft Cognitive Services Face API as you already see in PBI 1.

**Atrribute 1**

**Name:** emotionClient

**Type:** FaceClient

1. Create a new static property to store the API key, this key will be necessary to communicate with the Face API later the lab.
   1. The property must return the value of the class variable apiKey.

**Property 2**

**Name:** ApiKey

**Type:** string

1. Locate the *InitializeEmotionService* method and instantiate the *emotionClient* property passing to the constructor the ApiKey property. Because you are communicating with the API it is required to send a key to identify and validate calls to the API. Later in this lab you are going to get one key to test the application.
   1. Ex:

emotionClient = new FaceClient(

new ApiKeyServiceClientCredentials(ApiKey),

new System.Net.Http.DelegatingHandler[] { });

1. Locate the *ScoresToEmotionData* method and create a new variable of generic List<> of *EmotionData*. Examine the *EmotionAPI*\_Test example project created on PBI1 and create a new *EmotionData* object for each property and add it to the aforementioned list, the new *EmotionData* object will be assigned this way:

**EmotionData**

**EmotionName**: JSON element name

**EmotioScore**: JSON element value

Ex:

EmotionData data = new EmotionData();

data.EmotionName = "Fear";

data.EmotionScore = scores.Fear;

*Don’t forget return the List<> created at the method’s end.*

1. Locate the *RecognizeAsync* method and assign the result of an async call to the *emotionClient.Face.DetectWithStreamAsync* method, which is exposed by the client, to a var variable called result.
2. **Optional:** Locate the *RunTaskWithAutoRetryOnQuotaLimitExceededError* method. Declare a maximum retries and delay between retries and implement the call to the action parameter by coding a retry logic method. Because you can have transitory errors and, according to the pricing tier, request can start returning an *HttpStatus* 429 that means that you are sending too many requests to the API, it will be necessary to implement this method, so you can rely on the application to deal with this kind of situations and automatically retry for us the recognition.

**Configuration**

**retriesLeft**= 6

**delay**= 2

1. **Optional:** Modify your *RecognizeAsync* implementations by using the *RunTaskWithAutoRetryOnQuotaLimitExceededError* method so you can have a retry logic to deal with transitive errors
2. **Optional:** Change your *RunTaskWithAutoRetryOnQuotaLimitExceededError* so you can get an exponential retry feature.
3. **Mandatory**: Set the correct region for your Face API Uri in the *baseUri* string const

Task 4: Edit ImageAnalyzer class

1. Create a new public property by using the following values.

**Property 1**

**Name:** DetectedEmotion

**Type:** IList<Emotion>

1. Locate the *DetectEmotionAsync* method and implement it properly, inside the try block you will need to implement the following logic:

public async Task DetectEmotionAsync()

{

try

{

// Implement

}

catch (Exception e)

{

// Implement

}

finally

{

// Implement

}

}

* 1. Validate if the *GetImageStreamCallback* is not null and properly call the *EmotionServiceHelper.RecognizeAsync* class method and assign it’s result to the property created before.
  2. If *FilterOutSmallFaces* is true filter the *DetectedEmotion* property by calling the *IsFaceBigEnoughForDetection*.
     1. Tip: You can use *LINQ* to address this, have a look:

DetectedEmotion = DetectedEmotion.Where(f => CoreUtil.IsFaceBigEnoughForDetection(f.FaceRectangle.Height, this.DecodedImageHeight)).ToList();

1. Inside the catch block implement the following logic.
   1. Log the error by using the *ErrorTrackingHelper.TrackException* class method.
   2. Assign an empty emotion to *DetectedEmotion* property.
   3. If *ShowDialogOnFaceApiErrors* is true call the *ErrorTrackingHelper.GenericApiCallExceptionHandler* method by passing the exception and a custom message.
2. Inside the finally block call the *OnEmotionRecognitionCompleted* event.

Task 5: Build your project

1. Unload the *IntelligentKioskSample* project.
2. By this moment your project should be **compiling**. If not, try to fix issues.

# PBI 3: Setting up IntelligentKioskSample

#### Introduction

In this PBI, you will enhance the *IntelligentKiokSample* project to use the *IntelligentKioskSample.Azure.Api* project to capture an image with the computer camera identify the emotion scores with Face API.

#### Objectives

After completing this lab, you will be able to:

* Understand how Application Settings required to use Camera.

#### Prerequisites

You should finalize the PBI2 activities or checkout **PBI3-Begin** branch of AlpineSkiHouseHappinessMeter folder from GitHub

#### Scenario

To start using the services provided by the Microsoft Cognitive Services Vision API it will be necessary to add some basic references to work with this APIs in an easier way.

Task 1: Add missing Packages

1. Load the *IntelligentKioskSample* project.
2. Examine the source code of the *IntelligentKioskSample* project, if you try to compile it, there will be a lot of exception that you need take care of.
3. Use the NuGet Package Manager to lookup for *Microsoft.Azure.CognitiveServices.Vision.Face* package, install the latest version.

Install-Package Microsoft.Azure.CognitiveServices.Vision.Face -Version 2.0.0-preview

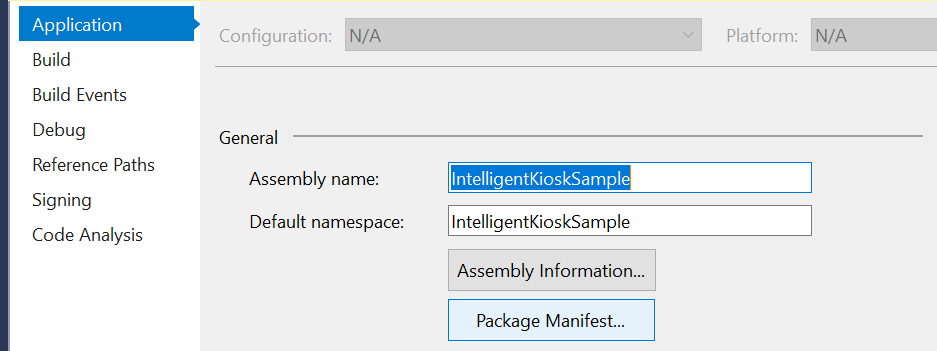
1. Now navigate thought project to the class file *EmotionServiceHelper* and add the required using to reference these libraries. Ex:

using Microsoft.Azure.CognitiveServices.Vision.Face;

using Microsoft.Azure.CognitiveServices.Vision.Face.Models;

Task 2: Adjust application settings

1. Open the *IntelligentKioskSample* project and locate the package manifest.



1. In the Application tab change the Display name and description of the application.

**Display Name:** Alpine Ski House

**Description:** Alpine Ski House Happiness Meter.

1. In the visual assets ensure the following values are set.

**Medium Tile:** Assets\squaretile-sdk.png

**App Icon:** Assets\squaretile-sdk-44.png

**Splash Screen:** Assets\splash-sdk.png

**Package Logo:** Assets\StoreLogo.png

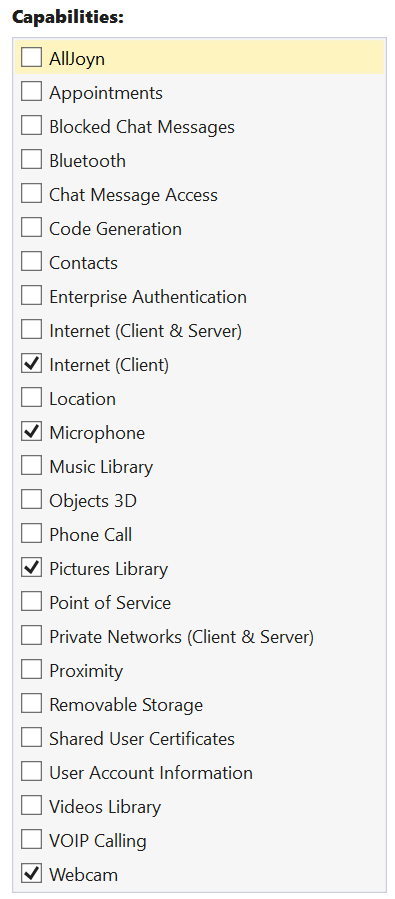
1. In the capabilities tab ensure the following options are checked.

**Internet (Client)**

**Microphone**

**Pictures Library**

**Web Cam**



1. In the Packaging tab, change the Package Name and Display Name

**Package Name:** AlpineSkiHouse

**Package Display Name:** AlpineSkiHouse Happiness Meter

**Publisher Display Name:** Your name

# PBI 4: Detecting Emotion in Image

#### Introduction

In this PBI, you will enhance the *IntelligentKiokSample* project to use the *IntelligentKioskSample.Azure.Api* library to capture an image with the computer camera identify the emotion scores with Face API.

#### Objectives

After completing this lab, you will be able to:

* Understand how to capture image from camera.
* Add emotion detection to application.

#### Prerequisites

You should finalize the PBI3 activities or checkout **PBI4-Begin** branch of AlpineSkiHouseHappinessMeter folder from GitHub

#### Scenario

This PBI will help you to Test your application and start capturing emotions.

Task 1: Run your application

1. Open the *IntelligentKioskSample* project and locate the package manifest.
2. Verify that your application runs and can access your camera and take a picture (it will capture it automatically), remember to add the API Key setting in the setting menu. Did the application work as expected? Did any emotion appear?
3. Debug your application and try to find if there is some missing implementation or method call. Tip: Locate the *PreviewImageFaces* method and review the *ImageWithFaceBorderUserControl* code to see if there is some missing method call.

Task 2: Wiring up Emotion Detection

1. Locate the *ImageWithFaceBorderUserControl*.*PreviewImageFaces* method and add call to *DetectAndShowEmotion* method if *ShowEmotionRecognition* is true.
2. Now, locate the *DetectAndShowEmotion* method implementation and in the *foreach* block you will need to create some logic:
   1. To show a rectangle with the detected face call the *faceUI.ShowFaceRectangle* method, using the data in emotion.FaceRectangle. Example:

faceUI.ShowFaceRectangle(emotion.FaceRectangle.Width \* renderedImageXTransform, emotion.FaceRectangle.Height \* renderedImageYTransform);

* 1. Set the Margin of the *faceUI.Margin* so it will appear aligned with the face and not in one of the corners of the screen. Example:

faceUI.Margin = new Thickness(

(emotion.FaceRectangle.Left \* renderedImageXTransform) + ((this.ActualWidth - this.imageControl.RenderSize.Width) / 2),

(emotion.FaceRectangle.Top \* renderedImageYTransform) + ((this.ActualHeight - this.imageControl.RenderSize.Height) / 2), 0, 0);

* 1. Set the *faceUI*.*BalloonBackground* and *faceUI*.*BalloonForeground* properties with the values in this.BalloonBackground and this.BalloonForeground.
  2. Show the emotions by calling the method *faceUI.ShowEmotionData(emotion)*.

1. Try your application again, remember to add the API Key setting in the setting menu.

Task 3: Add emoji Control

1. Locate the *EmotionEmojiControl* and implement the *UpdateEmotion* method.
2. In this case you will need to use the *ScoreToEmotionData* method declared on the *EmotionServiceHelper* class to determine the predominant emotion. Example:

EmotionData topEmotion = EmotionServiceHelper.ScoresToEmotionData(scores).OrderByDescending(d => d.EmotionScore).First();

1. Next implement a switch and set the label control to the emotion name, and the emoji text with the following values

**Anger**

Label = “Angry”, emoji = "\U0001f620"

**Contempt**

Label = "Contemptuous",emoji = "\U0001f612"

**Disgust**

Label = "Disgusted", emoji = "\U0001f627"

**Fear**

Label = "Afraid", emoji = "\U0001f628"

**Happiness**

Label = "Happy", emoji = "\U0001f60a"

**Neutral**

Label = "Neutral", emoji = "\U0001f614"

**Sadness**

label = "Sad", emoji = "\U0001f622"

**Surprise**

Label = "Surprised", emoji = "\U0001f632";

1. Now add the emoji control to the *ImageWithFaceBorderUserControl*  in the XAML file and add a call to the *UpdateEmotion(emotion.Scores)* method inside control’s C# code. Example:
   1. <local:EmotionEmojiControl x:Name="emotionEmojiControl" VerticalAlignment="Top" />
2. Try your application again and verify the emoji feature behavior.

END