Lesson 1.5

3.7.2020

Taking a look inside!

DAILY OBJECTIVE

In this lesson, we will break apart Wick Editor Azure projects that they previously interacted with. We'll explore and discuss the code, interactions, and APIs that make these projects function. We will also complete several challenges by remixing these projects.

DEFINITIONS

Here are a few terms that you might find useful today.

1. **API (Application Programming Interface):** A set of commands that are used to control specific interactions within a software program. For instance, in the examples provided, the Wick Editor projects utilize the "Azure Cognitive Services API" and provide a Face API, Text Analytics API, and an Image API.

Section 1: Understanding Project Structure

Each Wick Editor Azure demo project has several interactive parts that control the project, and allow us to process data.

In each example project, you'll see several objects outside of the main canvas.

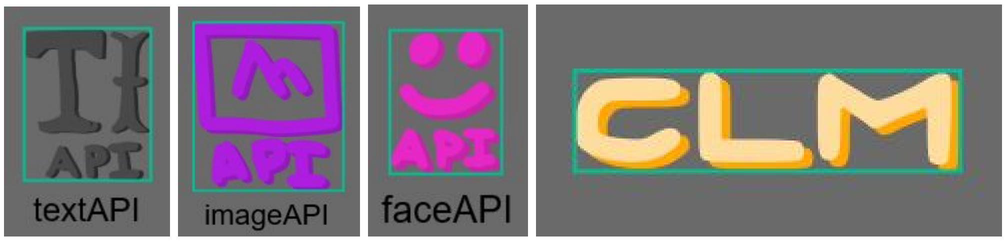
1.1 The API Objects

In each demo, these objects are explicitly needed. When creating your own project, be sure to copy these objects to your project or the example code provided will not work!

1. **The Azure Object**

   This object acts as the communication layer between our project and the Azure Cognitive Services system.

2. **The "custom" APIs**

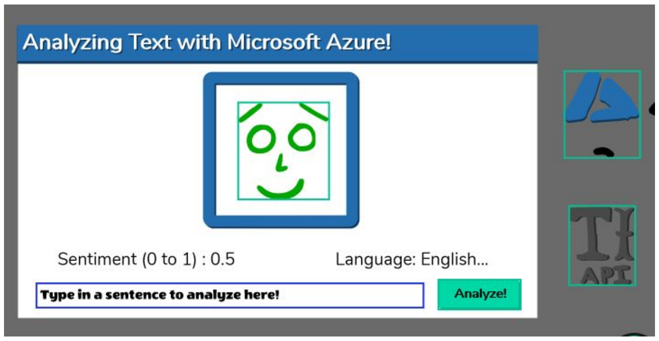
   Each project has its own API. Above, we show the textAPI, imageAPI, faceAPI and wick_clm objects. These APIs communicate between our project, and the Azure API!

Section 2: Breaking Down Examples

Now, we'll take a look at the examples we reviewed in lesson 1.3, and break them apart while remixing them! After reviewing each example, complete the remix challenges listed below.

2.1 The Text Analysis Demo

In this demo, students can analyze sentences and receive a sentiment analysis value from 0 to 1 representing negative to positive sentiment.

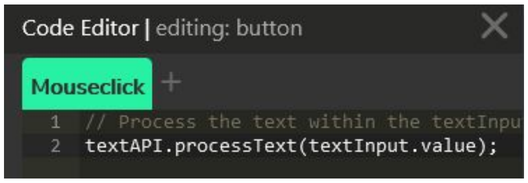

Analyzing Text with Microsoft Azure!
Sentiment (0 to 1) : 0.5
Language: English...
Type in a sentence to analyze here!
Analyze!
API

Direct Link	aka.ms/WE_TextAnalysisDemo_direct
Download Link	aka.ms/WE_TextAnalysisDemo

How does it work?

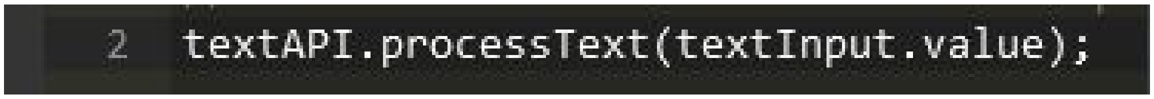
1. When students hit the analyze button, the project tells the textAPI to request a response from Azure Cognitive Services by using the "textAPI.processText(input)" command.

   This textAPI.processText() command is inside the analyze button's "mouseclick"script.



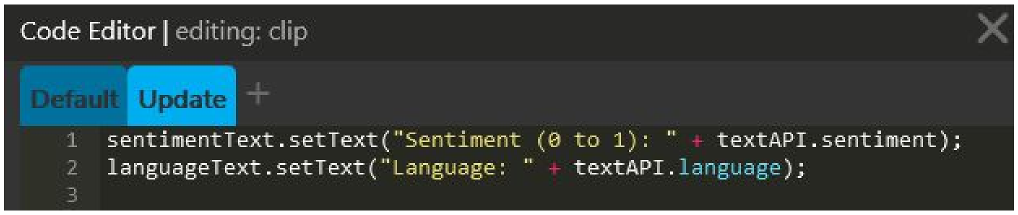
```
// Process the text within the textInpu
textAPI.processText(textInput.value);
```

   Note that we can get the value from the text box by using the textInput.value property.



```
textAPI.processText(textInput.value);
```

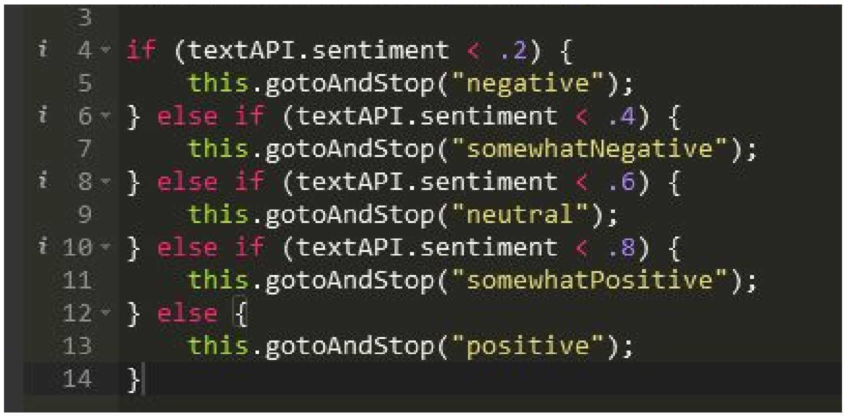
2. Next, when a response is received, the TextAPI updates it's "sentiment" and "language" properties. If we open the face object's "update" script, we can see where the changes occur.



```
sentimentText.setText("Sentiment (0 to 1): " + textAPI.sentiment);
languageText.setText("Language: " + textAPI.language);
```

These first two lines of code update the sentiment and language text objects visually!

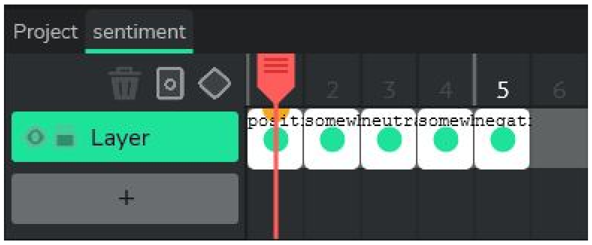
3. This next block of code changes the expression on the face shown to users.



```
if (textAPI.sentiment < .2) {
    this.gotoAndStop("negative");
} else if (textAPI.sentiment < .4) {
    this.gotoAndStop("somewhatNegative");
} else if (textAPI.sentiment < .6) {
    this.gotoAndStop("neutral");
} else if (textAPI.sentiment < .8) {
    this.gotoAndStop("somewhatPositive");
} else {
    this.gotoAndStop("positive");
}
```

Note that we use named frames here to control our character.

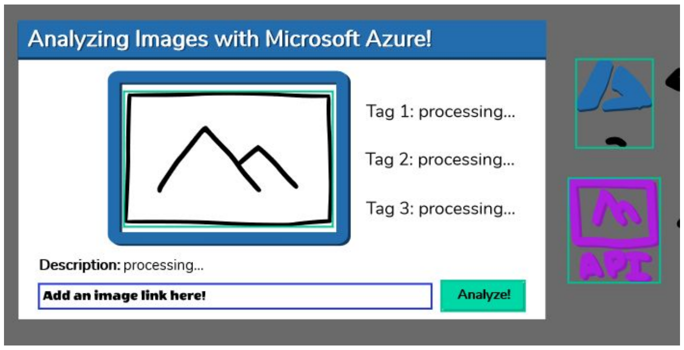
4. If we edit the timeline of the face object, we can see the 5 expression frames that make up the character.



You can see the full API for the TextAPI, including all of the properties and methods you're able to use, at the end of this guide!

2.2 The Image Analysis Demo

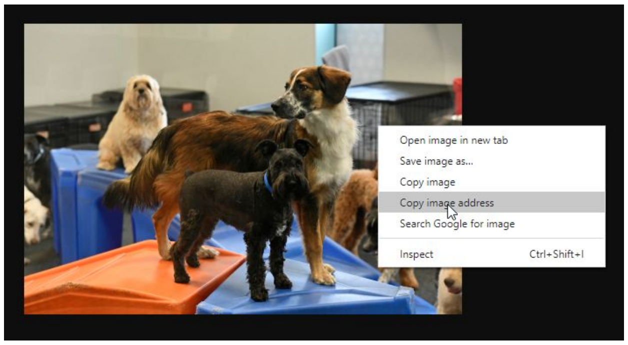
In this demo, students can analyze images from urls they input, and receive tags as well as AI generated descriptions of the image!



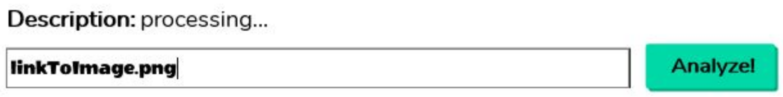
Direct Link	aka.ms/WE_ImageAnalysisDemo_direct
Download Link	aka.ms/WE_ImageAnalysisDemo

How does it work?

1. First, students find an image they'd like to analyze and copy that image's address by right-clicking, and using the Copy Image Address

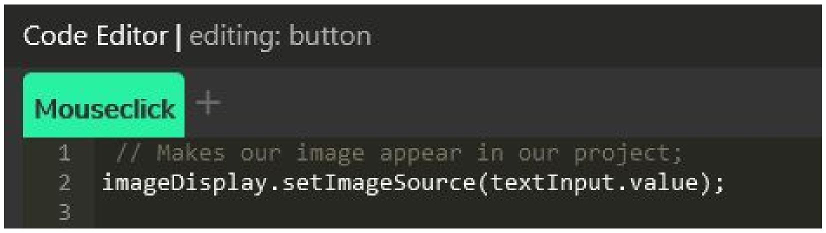


Then, play the project and paste this image link into the text input.



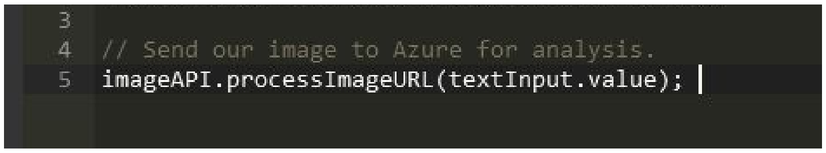
2. When students hit the "analyze" button. The project tells the imageAPI to request a response from Azure using the processImageURL() command.

In the analyze button's mouseclick script, the script displays the image to the user using the imageDisplay object.



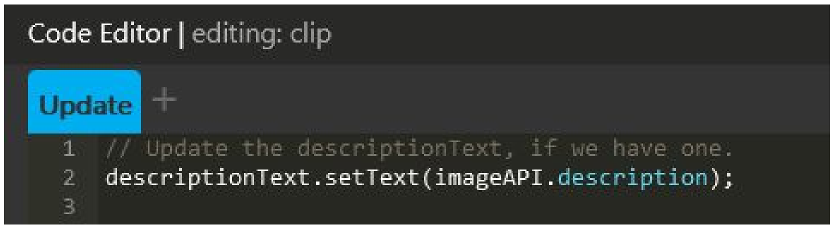
```
 // Makes our image appear in our project;
imageDisplay.setImageSource(textInput.value);
```

Then, we ask Azure to analyze the image with the imageAPI.processImageURL() command.



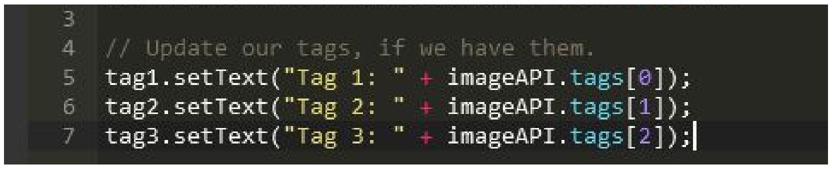
```
// Send our image to Azure for analysis.
imageAPI.processImageURL(textInput.value);
```

3. Once a response comes back from Azure Cognitive Services, the "update" script of the imageDisplay object sets the tags and description!



```
// Update the descriptionText, if we have one.
descriptionText.setText(imageAPI.description);
```

Tags are set by pulling them out of the tags Array, a list of all tags sent back by Azure Cognitive Services.



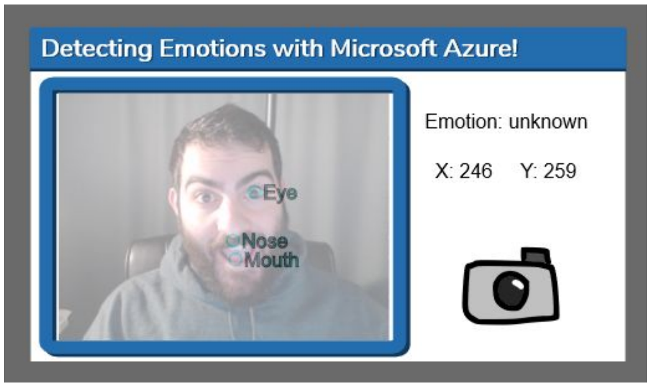
```
// Update our tags, if we have them.
tag1.setText("Tag 1: " + imageAPI.tags[0]);
tag2.setText("Tag 2: " + imageAPI.tags[1]);
tag3.setText("Tag 3: " + imageAPI.tags[2]);
```

4. Last note: In this example we only see 3 tags, but the imageAPI can return a much larger list of tags, usually around 15!

You can see the full API for the imageAPI, including all of the properties and methods you're able to use, at the end of this guide!

2.3 The Video Emotion Analysis Demo

In this demo, students can analyze an incoming video stream and determine what emotion a person is expressing, as well as where that person's face is on screen.



Emotion: unknown

X: 246 Y: 259

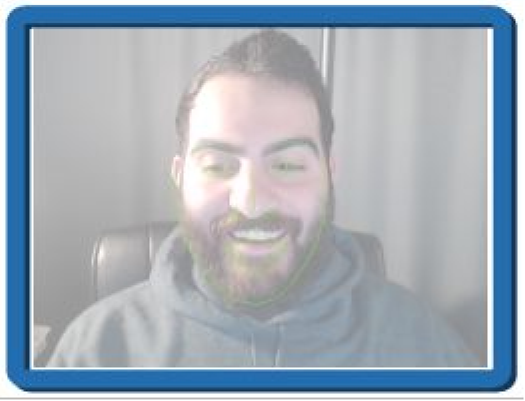
Direct Link	aka.ms/WE_VideoAnalysisDemo_direct
Download Link	aka.ms/WE_VideoAnalysisDemo

How does it work?

* This example uses two APIs, the CLM API for face position detection and the FaceAPI for facial emotion detection. We discuss how both work below!

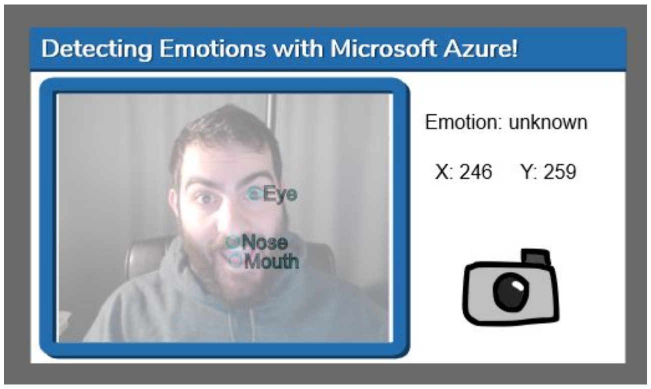
2.3 A - Face Position Detection (CLM)

1. As the camera captures video, this video is analyzed by the CLM API. You can tell the CLM API is working by the thin green outline around a user's face!

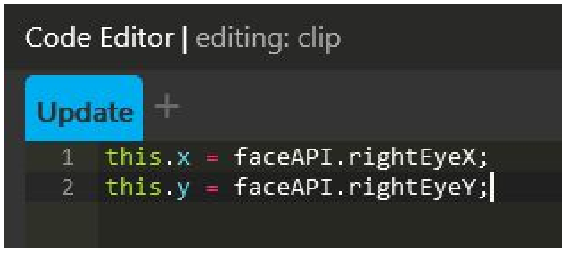


The closer this green outline matches your face, the more accurate the position data will be!

2. The CLM API is also being used to grab several different positions from a user's face, such as their eye positions, nose positions, and mouth positions.



You can see the code that controls these different markers by selecting one of the markers, and editing its "update" script.

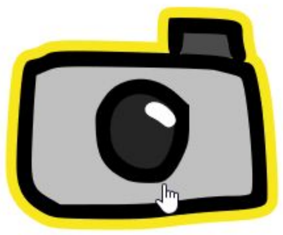


```
this.x = faceAPI.rightEyeX;
this.y = faceAPI.rightEyeY;
```

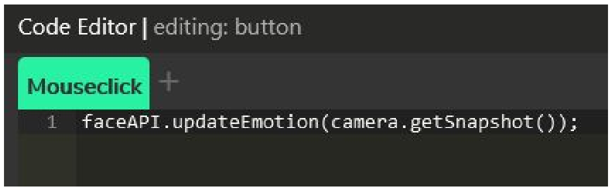
3. All facial points that students have access to can be found in the API docs at the end of this write up!

2.3 A - Emotion Detection (faceAPI + Azure)

1. When students hit the "Take Photo" button, the faceAPI asks azure to process the current frame using the faceAPI.updateEmotion() command.

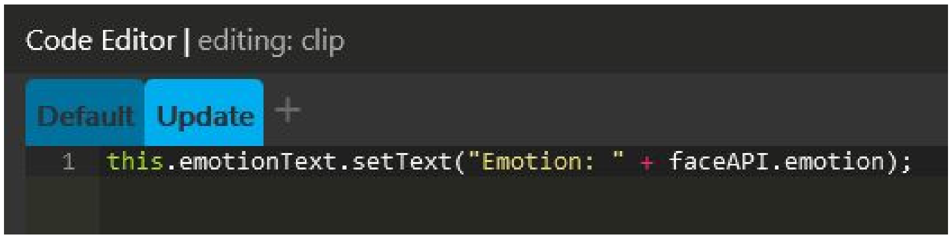


Open the button’s “mouseclick” script to see the code. Notice how we use “camera.snapshot()” to send a single image to the faceAPI’s updateEmotion() command.



```
faceAPI.updateEmotion(camera.getSnapshot());
```

2. Once a response comes back from Azure Cognitive Services, the ImageAPI’s “emotion” property is updated. Reviewing the sentiment object’s “update” script, we can see how one might use this!



```
this.emotionText.setText("Emotion: " + faceAPI.emotion);
```

Notice how this clip is referencing a text object inside of its own timeline!

Section 3: Challenges - Remixing Examples

With a better understanding of how these projects work, we'll start remixing the examples!

3.1 Remixing the Text Analysis Demo|

Remix the Text analysis demo and complete the following challenges

C 1

A new sentiment level

Add a new sentiment level that doesn't currently exist. For instance, can you add an "outraged" face for sentiments of exactly "0"?

C 2

Flipping sentiment levels

Flip the sentiment levels so the project responds opposite to the sentiment provided. For instance, a positive sentence should receive a negative face.

C 3

Bonus Challenge - Changing Frames

Make the project change to a different frame when a specific sentiment level is achieved. You may need to use the textAPI.textChanged property in an update script for this one.

Example Code:

```
// In an update script...
if (textAPI.textChanged) {
  // Do something here...
}
```

3.2 Remixing the Image Analysis Demo|

Remix the Image analysis demo and complete the following challenges

C 1

The responsive, cat detector.

Add a new clip that changes based on a specific tag appearing. For instance, creat a clip that displays a dog or cat, depending on if the tag “dog” or “cat” appears!

You may need to use the imageAPI.tagsChanged property in an update script for this one.

Example Code:

```
// In an update script...
if (imageAPI.tagsChanged) {
  // Do something here...
}
```

C 2

Frame Jumping

Edit the project so it brings you to a different frame based on if a specific tag is present.

C 3

Bonus Challenge - Multiple Tags

Create a project that changes to a different frame when two specific tags are present.

You may need to use the and (&&) operator inside an if statement for this one!

3.3 Remixing the Video Emotion Analysis Demo|

Remix the Video Emotion analysis demo and complete the following challenges

C 1	**Face Tracking** Add a new face tracking element that uses the API properties of the face. For instance, add a marker that tracks your left eye!
C 2	**Emotion Jumping** Edit the project so that the timeline changes based on the emotion returned. For instance, make the timeline move to frame 2 if the emotion is “sadness”.
C 3	**Bonus Challenge - Face Filter** Create a face filter that converts the user in front of the camera into an animal! Add at least one element for each eye, a nose tracker, and a mouth tracker!

PROCESS GUIDE

textAPI - Documentation

Use the following documentation to control projects using the textAPI.

With the textAPI and azure object in your project, you can use these values and functions:

Properties:

- **textAPI.sentiment** {number} A sentiment value representing the processed text. This value will be from 0 to 1, where 0 is very negative and 1 is very positive.
- **textAPI.language** {string} The language Azure believes this text is in.
- **textAPI.value** {string} The text that the textAPI is currently processing.
- **textAPI.textChanged** {boolean} true for 1 frame when the text has been updated and analyzed.

Methods:

- **textAPI.processText(text)** {function} Sends text to Azure Cognitive Services to be processed. The text passed in must be a string.

PROCESS GUIDE

imageAPI - Documentation

Use the following documentation to control projects using the imageAPI.

With the imageAPI and azure object in your project, you can use these values and functions:

Values:

- **imageAPI.description** {string} A string representing a computer generated description of the image passed to Azure Cognitive Services.
- **imageAPI.tags** {string[]} An array of strings representing likely tags for the image passed in. This can return any number of tags, including no tags at all.
- **imageAPI.imageChanged** {boolean} A boolean which is true for 1 frame when a new imageURL has been analyzed.

Methods:

- **imageAPI.processImageURL(url)** {function} sends an imageURL to Azure Cognitive Services for analysis.

PROCESS GUIDE

faceAPI - Documentation

Use the following documentation to control projects using the faceAPI.

With the CLM Library , faceAPI and Azure API in your project, you can use these values and functions:

Values:

- **faceAPI.emotion** {string} The most likely emotion of the face as returned by Azure! This value must be updated manually by calling faceAPI.update(). This value can be one of “anger”, “contempt”, “disgust”, “fear”, “happiness”, “neutral”, “sadness” or “surprise”.
- **faceAPI.leftEyeX** {string} The left eye’s x position (left to right).
- **faceAPI.leftEyeY** {string} The left eye’s y position (top to bottom).
- **faceAPI.rightEyeX** {string} The right eye’s x position (left to right).
- **faceAPI.rightEyeY** {string} The right eye’s y position (top to bottom).
- **faceAPI.noseX** {string} The nose’s x position (left to right).
- **faceAPI.noseY** {string} The nose’s y position (top to bottom).
- **faceAPI.mouthX** {string} The mouth’s x position based on its center point (left to right).
- **faceAPI.mouthY** {string} The mouth’s y position based on its center point (top to bottom).
- **faceAPI.faceX** {string} The face’s x position based on the center point of the face (left to right).
- **faceAPI.faceY** {string} The face’s y position based on the center point of the face (top to bottom).

Methods:

- faceAPI.updateEmotion(image) {function} Takes an image and updates the faceAPI with results from Azure Cognitive Services. Often takes camera.getSnapshot() as an input.