**Build a intelligent Video Gallery with Video Indexer**

Since around February, 2005 when YouTube first emerged, and even before that, individuals and corporations have been creating and consuming vast amounts of video, it’s the content king, it offers a strong way of expression than images or text and has become the preferred format for knowledge exchange. Video is great for learning. How many times have you found a solution to virtually any problem on either a free video site like Bing or YouTube, or pay sites like Lynda and Pluralsight. Those sites are successful for one reason: metadata. Without detailed metadata about the videos they host, there would be no way to find the content that will solve your problem.

So, while video is great for those who have a great system for managing metadata, for the rest of us it’s a real drag. Video without metadata is closed content, unstructured data. What we need is a way to turn this closed content into quality metadata. Wouldn’t it be great if we could not only generate video metadata automatically, but actually educe insights that are helpful to our bottom line?

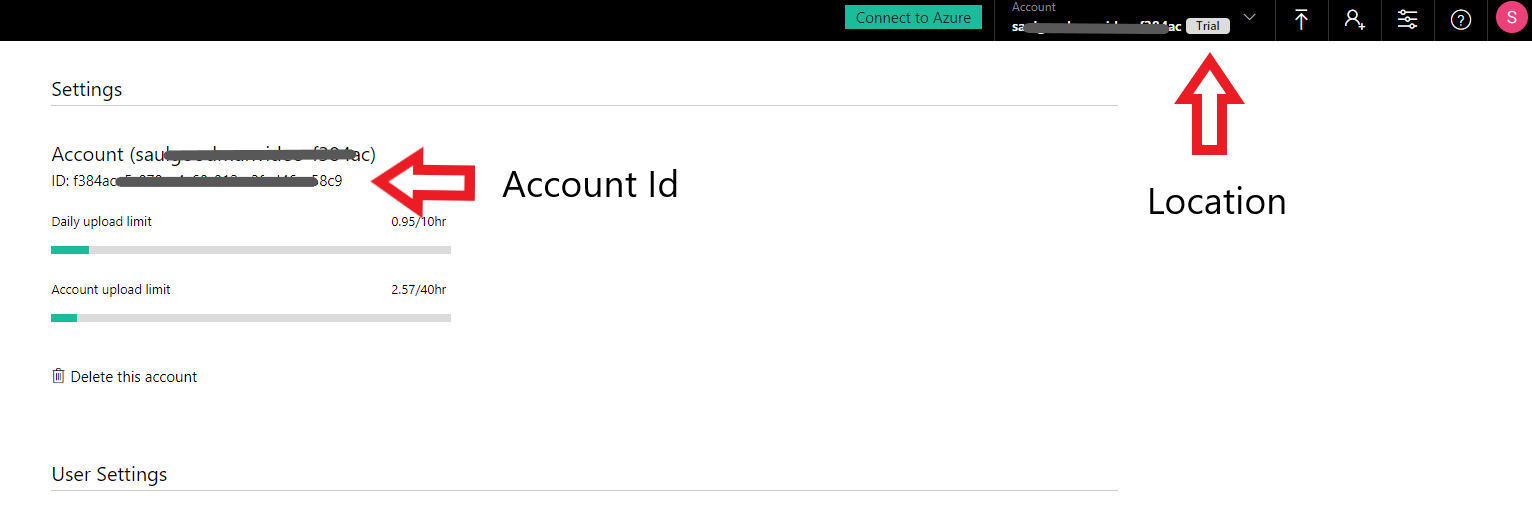
The Azure Video Indexer (VI) is an amazing technology that does just that. If you’re already an Azure customer or developer, the VI is just another REST service. You can call it to generate metadata, allowing you to gain insights into the content of your videos.

Imagine an intelligent video gallery where you can upload your videos or point to existing videos by URL. The Video Indexer discovers the insights and stores the metadata, letting you find trends, insights, and specific content.

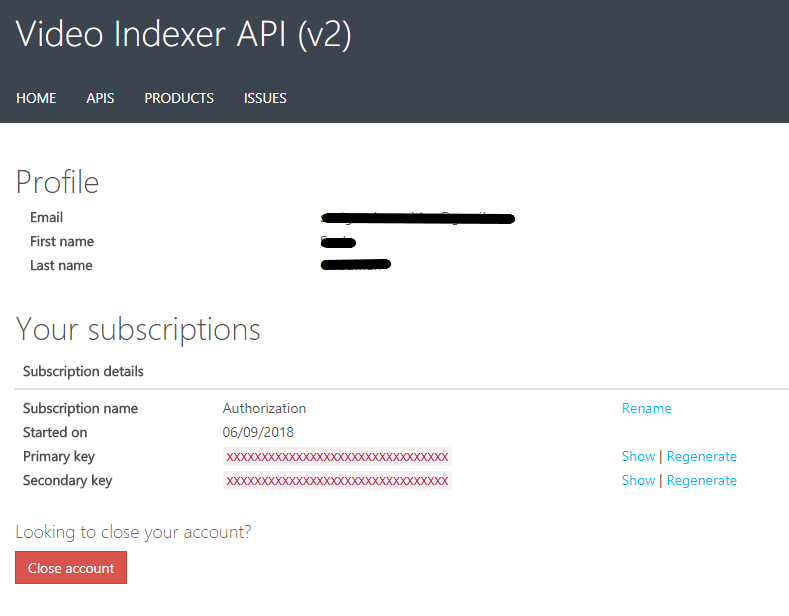
You can imagine a web page for uploading videos and specifying videos by URL, and then a dashboard for discovering insights about those videos. On the back end, we’re going to use the Video Indexer service-API to get those insights as metadata in JSON format, and store that metadata in a CosmosDB database.

Before we can get started we have to register an account with both the Video Indexing Service as well as the Video Indexer API. First you should register with the Video Indexing Service itself at<https://www.videoindexer.ai/>. Once your registration is complete, you will need to retrieve two key variables:

1. Your account Id (available under Settings)
2. Your account Location (this will appear in the top bar next to your login name. Be aware that for a Trial account, Trial is your location)



Once you have completed this, navigate to the Video Indexer API portal at<https://api-portal.videoindexer.ai/>, and subscribe using the same account. Upon successful login to the Indexer API portal, select the Products tab, Authorization and Subscribe. You will then be given a Primary and Secondary key; one of these keys also be required to access your video library in the Video Indexing Service.



At this point, you are ready to start coding. Before you can do any activity utilizing the Video Index service, you must obtain an Account Access Token. These tokens are good for one hour and can be easily retrieved with the sample code as shown below:

// THIS WILL BE UNIVERSAL TO ALL USERS

string *apiURL* = "https://api.videoindexer.ai";

//PRIMARY KEY OR SECONDARY KEY ACQUIRED WHEN YOU SUBSCRIBED TO THE VIDEO INDEX API

string *yourApiKey* = "93xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";

//ACCOUNT ID YOU RETRIEVED FROM THEH VIDEO INDEX SERVICE PORTAL

string *yourAccountId* = "f3xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

//THE LOCATION YOU RETRIEVED FROM THE VIDEO INDEX SERVICE PORTAL

string *yourLocation* = "Trial";

var *handler* = new HttpClientHandler();

*handler*.AllowAutoRedirect = false;

var *client* = new HttpClient(*handler*);

*client*.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", *yourApiKey*);

// obtain account access token

var *accountAccessTokenRequestResult* = *client*.GetAsync($"{*apiURL*}/auth/{*yourLocation*}/Accounts/{*yourAccountId*}/AccessToken?allowEdit=true").Result;

var *accountAccessToken* = *accountAccessTokenRequestResult*.Content.ReadAsStringAsync().Result.Replace("\"", "");

*client*.DefaultRequestHeaders.Remove("Ocp-Apim-Subscription-Key");

Once you have your Account Access Token, you upload the video with a simple HTTP post call, and that returns a VideoId. You can use that to get a Video Access Token. Note, that the Video Access Token may not be available immediately.

// We need a name and a URL for the video

string *videoName* = "Bill Gates at CES 2008";

string *videoURL* = "https://sec.ch9.ms/ch9/5227/b043bee1-8fc6-4ee2-9265-9e0f00b85227/BillGCES2008\_high\_ch9.mp4";

// This code starts the upload and immediately returns when uploaded

var *content* = new MultipartFormDataContent();

var *uploadRequestResult* = *client*.PostAsync($"{*apiURL*}/{*yourLocation*}/Accounts/{*yourAccountId*}/Videos?accessToken={*accountAccessToken*}&name={*videoName*}& description=some\_description&privacy=private&partition=some\_partition&videoUrl={*videoURL*}", *content*).Result;

// This code reads the upload result from the HTTP result

var *uploadResult* = *uploadRequestResult*.Content.ReadAsStringAsync().Result;

Console.WriteLine("Upload complete.");

// Now we just re-hydrate the JSON into an object and read the Video Id, which we

// can use to make subsequent API calls about this video

var *videoIdDesearialized* = JsonConvert.DeserializeObject<dynamic>(*uploadResult*)["id"];

string *videoId* = *videoIdDesearialized*.Value;

// OBTAIN VIDEO ACCESS TOKEN

var *videoAccesstoken* = "";

System.Net.ServicePointManager.SecurityProtocol =

System.Net.ServicePointManager.SecurityProtocol |

System.Net.SecurityProtocolType.Tls12;

*client*.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", *yourApiKey*);

var *videoTokenRequestResult* =

*client*.GetAsync($"{*apiURL*}/auth/{*yourLocation*}/Accounts/{*yourAccountId*}/Videos/{*videoId*}/AccessToken?allowEdit=true").Result;

*videoAccesstoken* =

*videoTokenRequestResult*.Content.ReadAsStringAsync().Result.Replace("\"", "");

*client*.DefaultRequestHeaders.Remove("Ocp-Apim-Subscription-Key");

Now you have to wait until the video has been completely indexed. Once again, you can poll for it using a simple HTTP call

var *videoGetIndexRequestResult* = *client*.GetAsync($"{apiURL}/{*yourLocation*}/Accounts/{*yourAccountId*}/Videos/{*videoId*}/Index?accessToken={*videoAccesstoken*}&language=English").Result;

var *videoGetIndexResult* = *videoGetIndexRequestResult*.Content.ReadAsStringAsync().Result;

var *processingState* = JsonConvert.DeserializeObject<dynamic>(*videoGetIndexResult*)["state"];

if (*processingState* != "Uploaded" && *processingState* != "Processing")

{

Console.WriteLine("Indexing complete.");

}

Once finished, you can use that VideoId retrieve all the data extracted by the Video Indexing service..

var *uri* = *apiURL* + "/" + *yourLocation* + "/Accounts/" + *yourAccountId* + "/Videos/" + *videoId* + "/Index?accessToken=" + *accountAccessToken* + "&language=English";

var task = client.GetAsync(*uri*)

.ContinueWith((*taskwithresponse*) =>

{

var *response* = *taskwithresponse*.Result;

var *jsonString* = response.Content.ReadAsStringAsync();

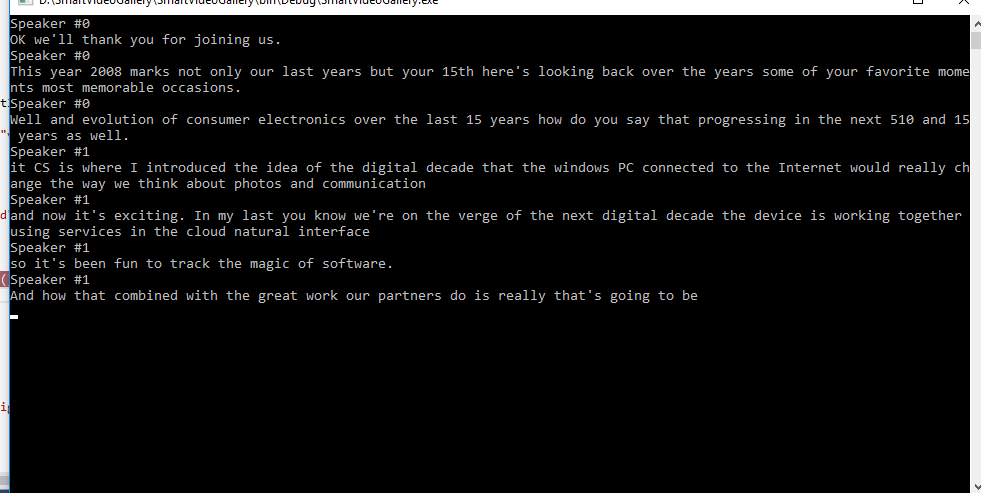
*jsonString*.Wait();

var *deserializedJson* = JObject.Parse(*jsonString*.Result);

});

task.Wait();

Now you can parse this data to get all sorts of cool insights...like a complete transcription:



var *transcripts* = *deserializedJson*["videos"][0]["insights"]["transcript"];

foreach (var *transcript* in *transcripts*)

{

string *transcriptSpeakerName* = string.Empty;

try

{

int *transcriptionSpeakerId* = Convert.ToInt32(*transcript*["speakerId"].ToString());

*transcriptSpeakerName* = *deserializedJson*["videos"][0]["insights"]["speakers"][*transcriptionSpeakerId*]["name"].ToString();

}

catch (Exception *NoSpeakerFound*)

{

*NoSpeakerFound*.Data.Clear();

*transcriptSpeakerName* = "Speaker not found";

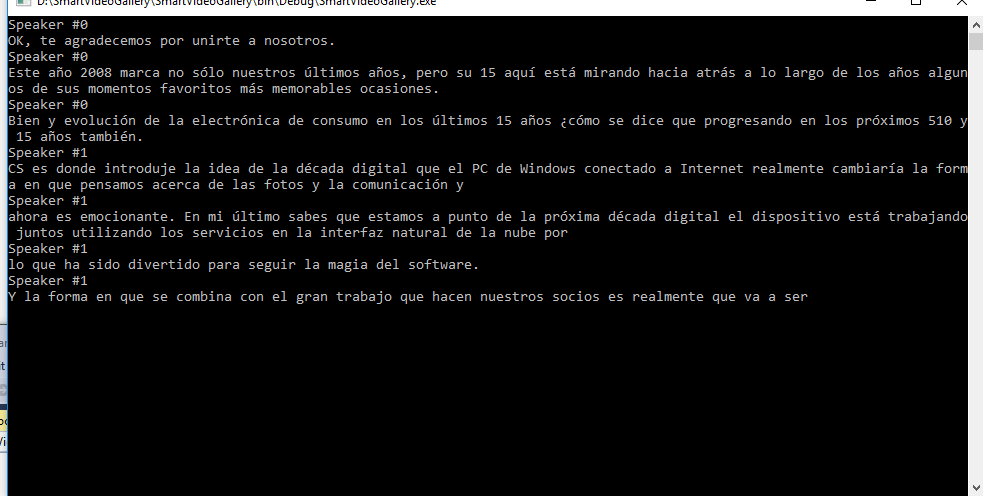
}

Console.WriteLine(*transcriptSpeakerName*);

Console.WriteLine(*transcript*["text"].ToString());

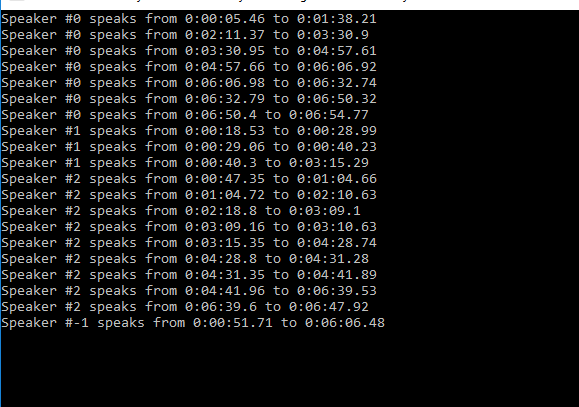
}

Want that in a different language? How about Spanish? Changing the language is as simple as changing the language specified in the URI.



var *uri* = *apiURL* + "/" + *yourLocation* + "/Accounts/" + *yourAccountId* + "/Videos/" + *videoId* + "/Index?accessToken=" + *accountAccessToken* + "&language=Spanish";

Let’s get a list of who spoke and when. That can be important when looking for a particular passage:

****

var *speakers* = *deserializedJson*["videos"][0]["insights"]["speakers"];

foreach (var *speaker* in *speakers*)

{

string *speakerName* = string.Empty;

*speakerName* = *speaker*["name"].ToString();

var *frequencyInstances* = *speaker*["instances"];

foreach (var *instance* in *frequencyInstances*)

{

string *beginTime* = instance["start"].ToString();

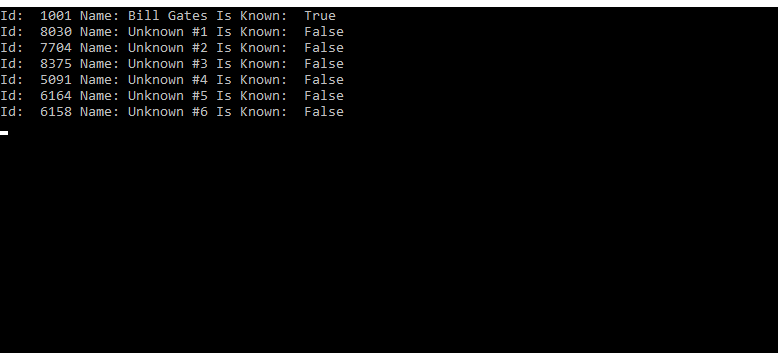
string *endTime* = instance["end"].ToString();

Console.WriteLine(*speakerName* + " speaks from " + *beginTime* + " to " + *endTime*);

}

}

If there are famous people in the video, like in our video, you can discover who they are:



var *faces* = *deserializedJson*["summarizedInsights"]["faces"];

foreach (var *face* in *faces*)

{

string *faceName* = *face*["name"].ToString();

bool *isKnown* = true;

try

{

if (*faceName*.Substring(0, 9) == "Unknown #")

{

isKnown = false;

}

}

catch (Exception *UnknownCompareException*)

{

*UnknownCompareException*.Data.Clear();

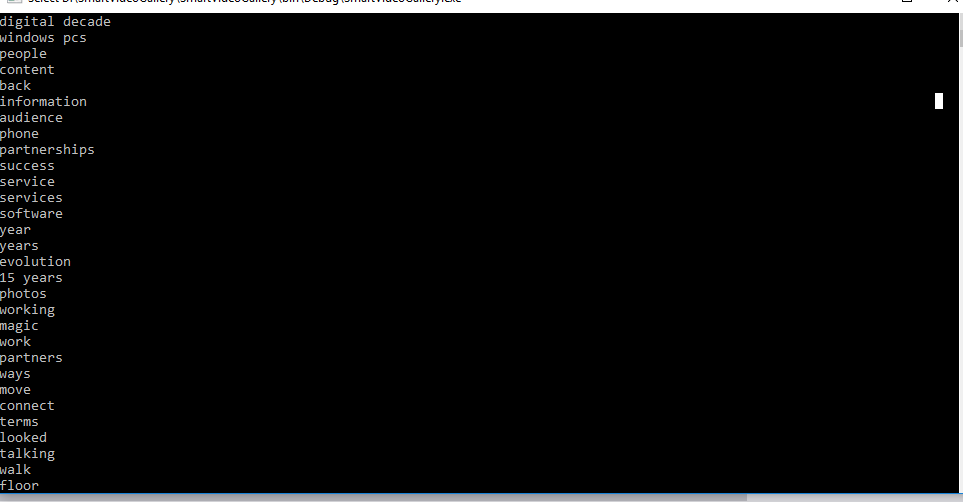
*isKnown* = true;

}

Console.WriteLine("Id: " + *face*["id"].ToString() + " Name: " + *faceName* + " Is Known: " + *isKnown*.ToString());

}

Keywords are an important part of video metadata. Who determines what they are? The Video Indexer does, and it does a pretty decent job of pulling out key phrases that can be searched on:



var *keyWords* = *deserializedJson*["summarizedInsights"]["keywords"];

foreach (var *keyword* in *keyWords*)

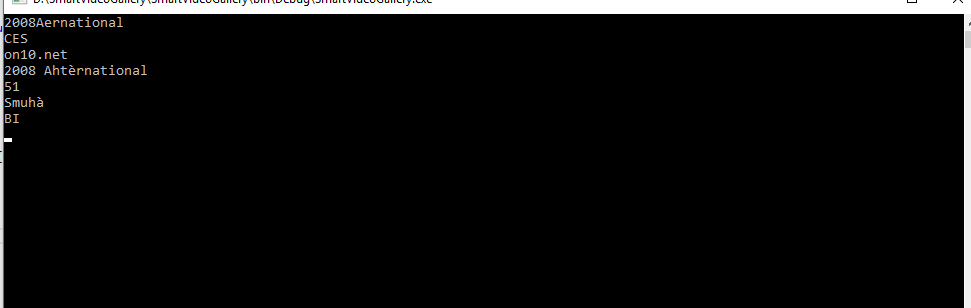
{

Console.WriteLine(*keyword*["name"].ToString());

}

Sometimes you don’t remember anything about the video but you remember an object either in the foreground or background. In our video there is clearly a man playing guitar on the TV in the background. Here’s how we can find it:

If there is any text on the screen, we’d like to capture that too, so we can search on it.



var *texts* = *deserializedJson*["videos"][0]["insights"]["ocr"];  
  
foreach (var *text* in *texts*)  
{  
 string *textFound* = text["text"].ToString();  
 Console.WriteLine(*textFound*);

}

In addition to text, we can get an idea of objects that exist within the video by polling for labels. Notice how the indexer notes there is a television in the video as well as a suit.



var *labels* = *deserializedJson*["summarizedInsights"]["labels"];

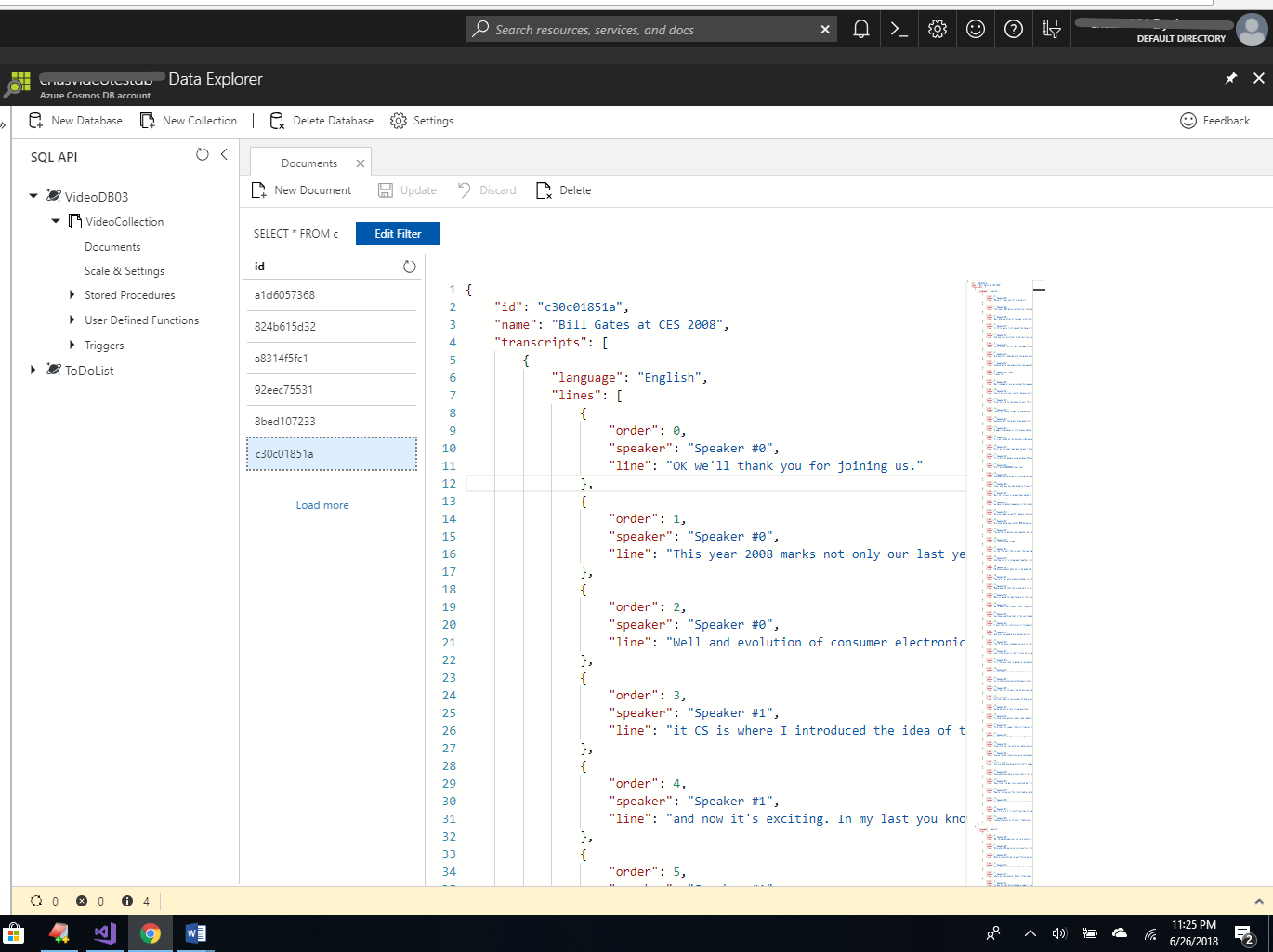
foreach (var *label* in *labels*)

{

Console.WriteLine((*label*["name"].ToString()));

}

Now that we have all this great data that we’ve pulled back from the Video Indexer, we need to store it so it can be queried later. For this we can use CosmosDB, an extremely low-latency scalable database platform in Microsoft Azure and store it as a Video Object type that we have defined.



public async Task AddVideoToCosmosDB(Video *videoObject*)

{

string *yourCosmosDbUri* = "https://xxxxxxxxxxxxxxxxxxx/";

string *yourCosmosKey* = "dixxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx==";

string *yourCosmosDbName* = "cosmosDbName";

string *yourCosmosDbCollection* = "cosmosCollectionName";

using (DocumentClient *documentClient* = new DocumentClient(new Uri(*yourCosmosDbUri*), *yourCosmosKey*))

{

await *documentClient*.CreateDatabaseIfNotExistsAsync(new Database { Id = *yourCosmosDbName* });

await *documentClient*.CreateDocumentCollectionIfNotExistsAsync(UriFactory.CreateDatabaseUri(*yourCosmosDbName*), new DocumentCollection { Id = *yourCosmosDbCollection* });

try

{

await *documentClient*.ReadDocumentAsync(UriFactory.CreateDocumentUri(*yourCosmosDbName*, *yourCosmosDbCollection*, *videoObject*.Id));

}

catch (DocumentClientException *de*)

{

if (*de*.StatusCode == HttpStatusCode.NotFound)

{

await *documentClient*.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(*yourCosmosDbName*, *yourCosmosDbCollection*), *videoObject*);

}

}

}

}

Once it’s in the database, it’s easy to find videos by any of the metadata we’ve saved. Here’s how you can find a video by it’s name:

using (DocumentClient *documentClient* = new DocumentClient(new Uri(*yourCosmosDbUri*),*yourCosmosDbUri*))

{

// Set some common query options

FeedOptions *queryOptions* = new FeedOptions { MaxItemCount = -1 };

// QUERY FOR VIDEOS BY NAME

IQueryable<Video> *videoQuery* = documentClient.CreateDocumentQuery<Video>(

UriFactory.CreateDocumentCollectionUri(*yourCosmosDbName*, *yourCosmosDbCollection*), *queryOptions*)

.Where(*v* => *v*.Name == "Bill Gates at CES 2008");

// The query is executed synchronously here, but can also be executed asynchronously via the IDocumentQuery<T> interface

Console.WriteLine("Running LINQ query...");

foreach (Video *video* in *videoQuery*)

{

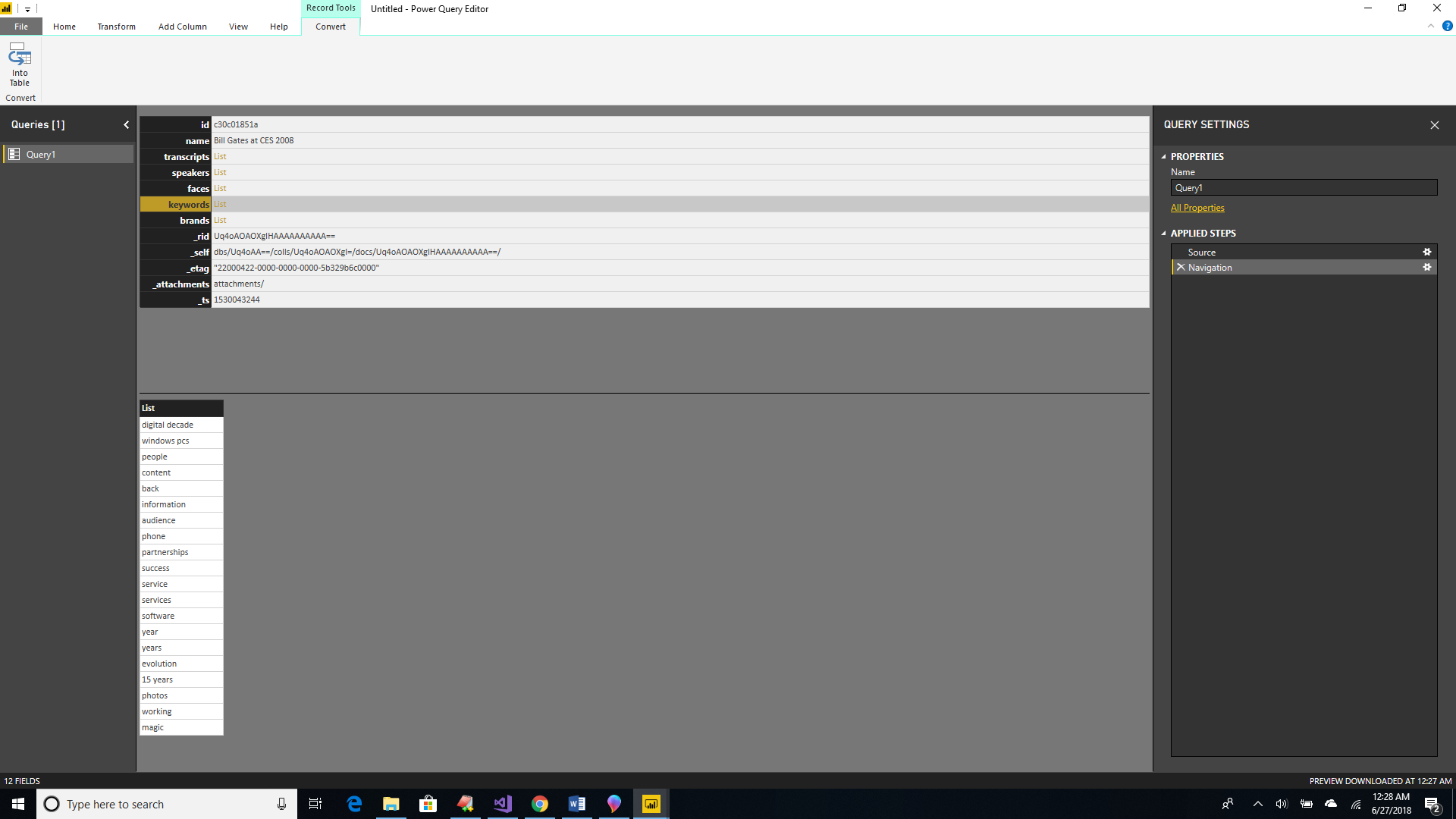
Console.WriteLine("");

Console.WriteLine(JsonConvert.SerializeObject(*video*));

}

}

Now let’s really kick it up a notch and connect a Power BI dashboard to our database. This will let us gain powerful insights into the video content



Video Indexer turns static closed videos into discoverable information, opening up a world of value to anyone who can use a web browser. The API is so simple, any developer can call into it on any platform. Video Indexer facilitates the delivery of powerful insights for your end users. The sky’s the limit!