

Contents

Computer Vision Documentation

Overview

[What is Computer Vision?](#)

Quickstarts

[Summary](#)

C#

[Analyze an image](#)

[Generate a thumbnail](#)

[Extract printed text](#)

[Extract handwritten text](#)

cURL

[Analyze a remote image](#)

[Analyze a local image](#)

[Generate a thumbnail](#)

[Extract printed text](#)

Java

[Analyze an image](#)

[Generate a thumbnail](#)

[Extract printed text](#)

[Extract handwritten text](#)

JavaScript

[Analyze an image](#)

[Generate a thumbnail](#)

[Extract printed text](#)

[Extract handwritten text](#)

Node.js

[Analyze an image](#)

[Generate a thumbnail](#)

[Extract printed text](#)

PHP

- [Analyze an image](#)
- [Use a domain model](#)
- [Generate a thumbnail](#)
- [Extract printed text](#)

Python

- [Analyze a remote image](#)
- [Analyze a local image](#)
- [Use a domain model](#)
- [Generate a thumbnail](#)
- [Extract printed text](#)
- [Extract handwritten text](#)

Ruby

- [Analyze an image](#)
- [Generate a thumbnail](#)
- [Extract printed text](#)

How-to guides

First Computer Vision apps

- [C#](#)
- [Java](#)
- [JavaScript](#)
- [Python](#)

Obtain subscription keys

Call the Computer Vision API

Analyze videos in real time

Work with Visual Studio

- [Use the Computer Vision Connected Service](#)

Reference

Computer Vision API v2.0

Computer Vision API v1.0

SDKs

- [Go](#)

[Node.js](#)

[Python](#)

[Windows](#)

[Android](#)

[Swift](#)

[Resources](#)

[Samples](#)

[FAQ](#)

[Pricing](#)

[UserVoice](#)

[Stack Overflow](#)

[Azure Roadmap](#)

[Category taxonomy](#)

What is Computer Vision API Version 2.0?

5/25/2018 • 9 minutes to read • [Edit Online](#)

NOTE

For the previous version of Computer Vision API see:

- [Overview](#)
- [Computer Vision API Version 1.0](#)

The cloud-based Computer Vision API provides developers with access to advanced algorithms for processing images and returning information. By uploading an image or specifying an image URL, Microsoft Computer Vision algorithms can analyze visual content in different ways based on inputs and user choices. With the Computer Vision API, you can analyze images to:

- [Tag images based on content](#)
- [Categorize images](#)
- [Identify the type and quality of images](#)
- [Detect human faces and return their coordinates](#)
- [Recognize domain-specific content](#)
- [Generate descriptions of the content](#)
- [Use optical character recognition to identify printed text found in images](#)
- [Recognize text](#)
- [Distinguish color schemes](#)
- [Flag adult content](#)
- [Crop photos to be used as thumbnails](#)

Requirements

- Supported input methods: Raw image binary in the form of an application/octet stream or image URL.
- Supported image formats: JPEG, PNG, GIF, BMP.
- Image file size: Less than 4 MB.
- Image dimension: Greater than 50 x 50 pixels.

Tagging images

Computer Vision API returns tags based on more than 2000 recognizable objects, living beings, scenery, and actions. When tags are ambiguous or not common knowledge, the API response provides 'hints' to clarify the meaning of the tag in context of a known setting. Tags are not organized as a taxonomy and no inheritance hierarchies exist. A collection of content tags forms the foundation for an image 'description' displayed as human readable language formatted in complete sentences. Note, that at this point English is the only supported language for image description.

After uploading an image or specifying an image URL, Computer Vision API's algorithms output tags based on the objects, living beings, and actions identified in the image. Tagging is not limited to the main subject, such as a person in the foreground, but also includes the setting (indoor or outdoor), furniture, tools, plants, animals, accessories, gadgets etc.

Example



Returned Json

```
{
  'tags':[
    {
      "name": "grass",
      "confidence": 0.999999761581421
    },
    {
      "name": "outdoor",
      "confidence": 0.999970674514771
    },
    {
      "name": "sky",
      "confidence": 0.999289751052856
    },
    {
      "name": "building",
      "confidence": 0.996463239192963
    },
    {
      "name": "house",
      "confidence": 0.992798030376434
    },
    {
      "name": "lawn",
      "confidence": 0.822680294513702
    },
    {
      "name": "green",
      "confidence": 0.641222536563873
    },
    {
      "name": "residential",
      "confidence": 0.314032256603241
    },
  ],
}
```

Categorizing images

In addition to tagging and descriptions, Computer Vision API returns the taxonomy-based categories defined in previous versions. These categories are organized as a taxonomy with parent/child hereditary hierarchies. All categories are in English. They can be used alone or with our new models.

The 86-category concept

Based on a list of 86 concepts seen in the following diagram, visual features found in an image can be categorized ranging from broad to specific. For the full taxonomy in text format, see [Category Taxonomy](#).

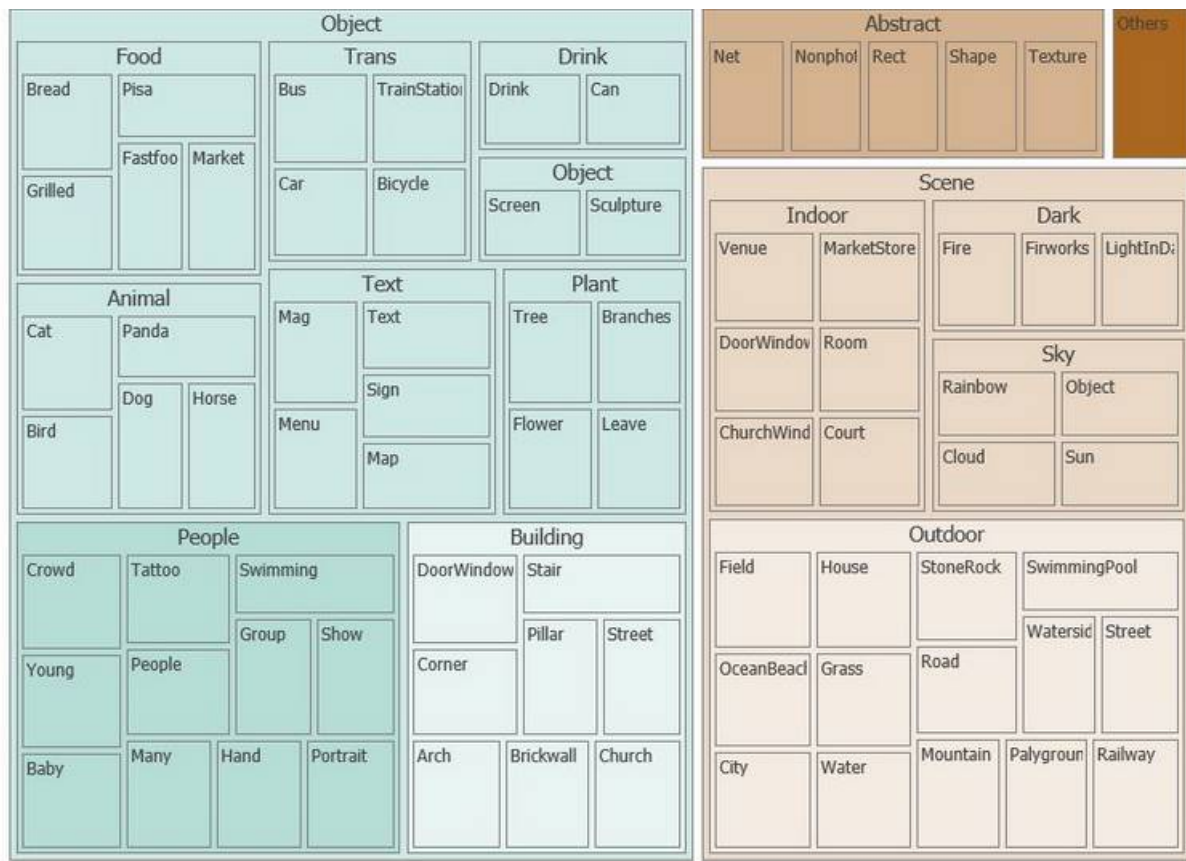







IMAGE	RESPONSE
	people
	people_crowd

IMAGE	RESPONSE
	animal_dog
	outdoor_mountain
	food_bread



Identifying image types

There are several ways to categorize images. Computer Vision API can set a boolean flag to indicate whether an image is black and white or color. The API can also set a flag to indicate whether an image is a line drawing or not. It can also indicate whether an image is clip art or not and indicate its quality on a scale of 0-3.

Clip-art type



Detects whether an image is clip art or not.

VALUE	MEANING
0	Non-clip-art
1	ambiguous
2	normal-clip-art
3	good-clip-art

IMAGE	RESPONSE
	3 good-clip-art
	0 Non-clip-art

Line drawing type



Detects whether an image is a line drawing or not.

IMAGE	RESPONSE
	True
	False

Faces

Detects human faces within a picture and generates the face coordinates, the rectangle for the face, gender, and age. These visual features are a subset of metadata generated for face. For more extensive metadata generated for faces

(facial identification, pose detection, and more), use the Face API.

IMAGE	RESPONSE
	[{ "age": 23, "gender": "Female", "faceRectangle": { "left": 1379, "top": 320, "width": 310, "height": 310 } }]
	[{ "age": 11, "gender": "Male", "faceRectangle": { "left": 113, "top": 314, "width": 222, "height": 222 } }, { "age": 11, "gender": "Female", "faceRectangle": { "left": 1200, "top": 632, "width": 215, "height": 215 } }, { "age": 41, "gender": "Male", "faceRectangle": { "left": 514, "top": 223, "width": 205, "height": 205 } }, { "age": 37, "gender": "Female", "faceRectangle": { "left": 1008, "top": 277, "width": 201, "height": 201 } }]

Domain-specific content

In addition to tagging and top-level categorization, Computer Vision API also supports specialized (or domain-specific) information. Specialized information can be implemented as a standalone method or with the high-level categorization. It functions as a means to further refine the 86-category taxonomy through the addition of domain-specific models.

Currently, the only specialized information supported are celebrity recognition and landmark recognition. They are domain-specific refinements for the people and people group categories, and landmarks around the world.

There are two options for using the domain-specific models:

Option one - scoped analysis

Analyze only a chosen model by invoking an HTTP POST call. If you know which model you want to use, specify the model's name. You only get information relevant to that model. For example, you can use this option to only look for celebrity-recognition. The response contains a list of potential matching celebrities, accompanied by their confidence scores.

Option two - enhanced analysis

Analyze to provide additional details related to categories from the 86-category taxonomy. This option is available for use in applications where users want to get generic image analysis in addition to details from one or more domain-specific models. When this method is invoked, the 86-category taxonomy classifier is called first. If any of the categories match that of known/matching models, a second pass of classifier invocations follows. For example, if 'details=all' or "details" include 'celebrities', the method calls the celebrity classifier after the 86-category classifier is called. The result includes tags starting with 'people_'.

Generating descriptions

Computer Vision API's algorithms analyze the content in an image. This analysis forms the foundation for a 'description' displayed as human-readable language in complete sentences. The description summarizes what is

found in the image. Computer Vision API's algorithms generate various descriptions based on the objects identified in the image. The descriptions are each evaluated and a confidence score generated. A list is then returned ordered from highest confidence score to lowest. An example of a bot that uses this technology to generate image captions can be found [here](#).

Example description generation






Returned Json

```
'description':{
  "captions":[
    {
      "type":"phrase",
      'text':'a black and white photo of a large city',
      'confidence':0.607638706850331
    }
  ]
  "captions":[
    {
      "type":"phrase",
      'text':'a photo of a large city',
      'confidence':0.577256764264197
    }
  ]
  "captions":[
    {
      "type":"phrase",
      'text':'a black and white photo of a city',
      'confidence':0.538493271791207
    }
  ]
  'description':[
    "tags":{
      "outdoor",
      "city",
      "building",
      "photo",
      "large",
    }
  ]
}
```

Perceiving color schemes

The Computer Vision algorithm extracts colors from an image. The colors are analyzed in three different contexts: foreground, background, and whole. They are grouped into 12 dominant accent colors. Those accent colors are black, blue, brown, gray, green, orange, pink, purple, red, teal, white, and yellow. Depending on the colors in an

image, simple black and white, or accent colors may be returned in hexadecimal color codes.

IMAGE	FOREGROUND	BACKGROUND	COLORS
	Black	Black	White
	Black	White	White, Black, Green
	Black	Black	Black

Accent color

Color extracted from an image designed to represent the most eye-popping color to users via a mix of dominant colors and saturation.






IMAGE	RESPONSE
	#BC6F0F
	#CAA501

IMAGE	RESPONSE
	#484B83

Black & white

Boolean flag that indicates whether an image is black&white or not.

IMAGE	RESPONSE
	True
	False

Flagging adult content

Among the various visual categories is the adult and racy group, which enables detection of adult materials and restricts the display of images containing sexual content. The filter for adult and racy content detection can be set on a sliding scale to accommodate the user's preference.

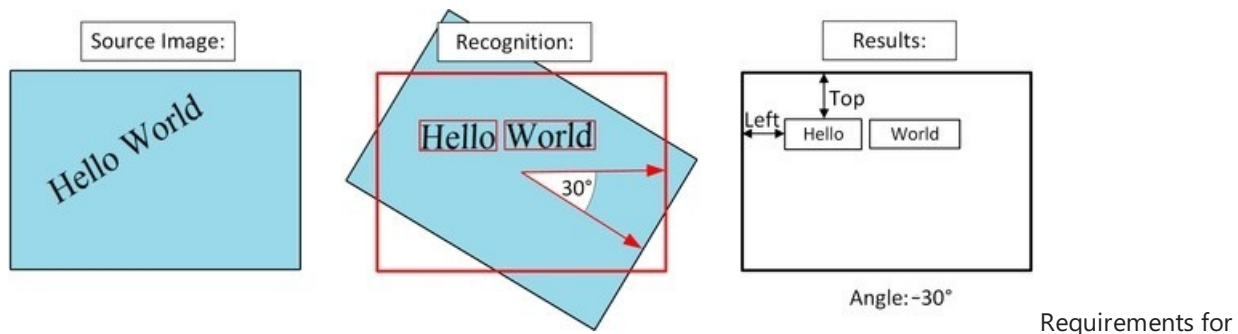
Optical character recognition (OCR)

OCR technology detects text content in an image and extracts the identified text into a machine-readable character stream. You can use the result for search and numerous other purposes like medical records, security, and banking. It automatically detects the language. OCR saves time and provides convenience for users by allowing them to take

photos of text instead of transcribing the text.

OCR supports 25 languages. These languages are: Arabic, Chinese Simplified, Chinese Traditional, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Romanian, Russian, Serbian (Cyrillic and Latin), Slovak, Spanish, Swedish, and Turkish.

If needed, OCR corrects the rotation of the recognized text, in degrees, around the horizontal image axis. OCR provides the frame coordinates of each word as seen in below illustration.



OCR:

- The size of the input image must be between 40 x 40 and 3200 x 3200 pixels.
- The image cannot be bigger than 10 megapixels.

The input image can be rotated by any multiple of 90 degrees plus a small angle of up to '40 degrees.

The accuracy of text recognition depends on the quality of the image. An inaccurate reading may be caused by the following situations:

- Blurry images.
- Handwritten or cursive text.
- Artistic font styles.
- Small text size.
- Complex backgrounds, shadows, or glare over text or perspective distortion.
- Oversized or missing capital letters at the beginnings of words
- Subscript, superscript, or strikethrough text.

Limitations: On photos where text is dominant, false positives may come from partially recognized words. On some photos, especially photos without any text, precision can vary a lot depending on the type of image.

Recognize text

This technology allows you to detect and extract printed or handwritten text from images of various objects with different surfaces and backgrounds, such as receipts, posters, business cards, letters, and whiteboards.

Text recognition saves time and effort. You can be more productive by taking an image of text rather than transcribing it. Text recognition makes it possible to digitize notes. This digitization allows you to implement quick and easy search. It also reduces paper clutter.

Input requirements:

- Supported image formats: JPEG, PNG, and BMP.
- Image file size must be less than 4 MB.
- Image dimensions must be at least 40 x 40, at most 3200 x 3200.

Note: this technology is currently in preview and is only available for English text.

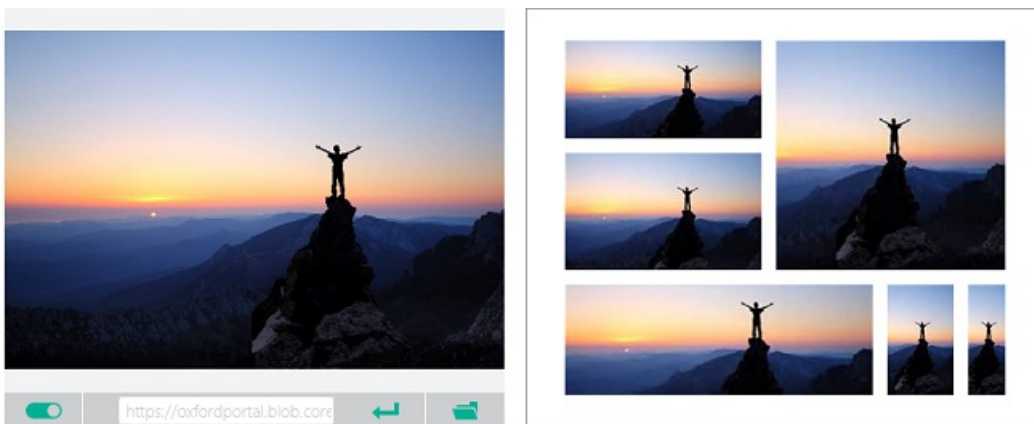
Generating thumbnails

A thumbnail is a small representation of a full-size image. Varied devices such as phones, tablets, and PCs create a need for different user experience (UX) layouts and thumbnail sizes. Using smart cropping, this Computer Vision API feature helps solve the problem.

After uploading an image, a high-quality thumbnail gets generated and the Computer Vision API algorithm analyzes the objects within the image. It then crops the image to fit the requirements of the 'region of interest' (ROI). The output gets displayed within a special framework as seen in below illustration. The generated thumbnail can be presented using an aspect ratio that is different from the aspect ratio of the original image to accommodate a user's needs.

The thumbnail algorithm works as follows:

1. Removes distracting elements from the image and recognizes the main object, the 'region of interest' (ROI).
2. Crops the image based on the identified region of interest.
3. Changes the aspect ratio to fit the target thumbnail dimensions.



Quickstart: Summary

6/11/2018 • 2 minutes to read • [Edit Online](#)

These quickstarts provide information and code samples to help you quickly get started using the Computer Vision API to accomplish the following tasks:

- Analyze a remote image
- Analyze a local image
- Detect celebrities and landmarks (domain models)
- Intelligently generate a thumbnail
- Detect and extract printed text (OCR) from an image
- Detect and extract handwritten text from an image

The code in each sample is similar. However, they highlight different Computer Vision features along with different techniques for exchanging data with the service, such as:

- *Generate a thumbnail* returns an image as a byte array in the body of the response.
- *Analyze a local image* requires the image to be included in the request as a byte array.
- *Extract handwritten text* requires two calls to retrieve the text.

Summary

QUICKSTART	REQUEST PARAMETERS	RESPONSE
Analyze a remote image	visualFeatures=Categories,Description,Color	JSON string
Analyze a local image	data=image_data (byte array)	JSON string
Detect celebrities	model=celebrities	JSON string
Generate a thumbnail	width=200&height=150&smartCropping=true	byte array
Extract printed text	language=unk&detectOrientation=true	JSON string
Extract handwritten text	handwriting=true	URL, JSON string*

* Two API calls are required. The first call returns a URL, which is used by the second call to get the text.

To rapidly experiment with the Computer Vision APIs, try the [Open API testing console](#).

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Analyze a local image with C#

6/15/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you analyze a local image to extract visual features using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze Image request

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Create a new Visual C# Console App in Visual Studio.
2. Install the Newtonsoft.Json NuGet package.
 - a. On the menu, click **Tools**, select **NuGet Package Manager**, then **Manage NuGet Packages for Solution**.
 - b. Click the **Browse** tab, and in the **Search** box type "Newtonsoft.Json".
 - c. Select **Newtonsoft.Json** when it displays, then click the checkbox next to your project name, and **Install**.
3. Replace `Program.cs` with the following code.
4. Replace `<Subscription Key>` with your valid subscription key.
5. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
6. Run the program.
7. At the prompt, enter the path to a local image.

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Replace <Subscription Key> with your valid subscription key.
        const string subscriptionKey = "<Subscription Key>";

        // You must use the same region in your REST call as you used to
        // get your subscription keys. For example, if you got your
```



```

// subscription keys from westus, replace "westcentralus" in the URL
// below with "westus".
//
// Free trial subscription keys are generated in the westcentralus region.
// If you use a free trial subscription key, you shouldn't need to change
// this region.
const string uriBase =
    "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze";

static void Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Analyze an image:");
    Console.Write("Enter the path to the image you wish to analyze: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Make the REST API call.
        Console.WriteLine("\nWait a moment for the results to appear.\n");
        MakeAnalysisRequest(imageFilePath).Wait();
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets the analysis of the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file to analyze.</param>
static async Task MakeAnalysisRequest(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameters. A third optional parameter is "details".
        string requestParameters =
            "visualFeatures=Categories,Description,Color";

        // Assemble the URI for the REST API Call.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Request body. Posts a locally stored JPEG image.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses content type "application/octet-stream".
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");

            // Make the REST API call.
            response = await client.PostAsync(uri, content);
        }
    }
}

```

```

        // Get the JSON response.
        string contentString = await response.Content.ReadAsStringAsync();

        // Display the JSON response.
        Console.WriteLine("\nResponse:\n\n{0}\n",
            JToken.Parse(contentString).ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("\n" + e.Message);
    }
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

Analyze Image response

A successful response is returned in JSON, for example:

```
{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "others_",
      "score": 0.0234375
    },
    {
      "name": "outdoor_",
      "score": 0.00390625
    }
  ],
  "description": {
    "tags": [
      "road",
      "building",
      "outdoor",
      "street",
      "night",
      "black",
      "city",
      "white",
      "light",
      "sitting",
      "riding",
      "man",
      "side",
      "empty",
      "rain",
      "corner",
      "traffic",
      "lit",
      "hydrant",
      "stop",
      "board",
      "parked",
      "bus",
      "tall"
    ],
    "captions": [
      {
        "text": "a close up of an empty city street at night",
        "confidence": 0.7965622853462756
      }
    ]
  },
  "requestId": "ddd1ac9-7e66-4c47-bdef-222f3fe5aa23",
  "metadata": {
    "width": 3733,
    "height": 1986,
    "format": "Jpeg"
  },
  "color": {
    "dominantColorForeground": "Black",
    "dominantColorBackground": "Black",
    "dominantColors": [
      "Black",
      "Grey"
    ],
    "accentColor": "666666",
    "isBwImg": true
  }
}
```

Next steps

Explore a basic Windows application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Use Computer Vision with C#](#)

Quickstart: Generate a thumbnail with C#

6/15/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Create a new Visual C# Console App in Visual Studio.
2. Install the Newtonsoft.Json NuGet package.
 - a. On the menu, click **Tools**, select **NuGet Package Manager**, then **Manage NuGet Packages for Solution**.
 - b. Click the **Browse** tab, and in the **Search** box type "Newtonsoft.Json".
 - c. Select **Newtonsoft.Json** when it displays, then click the checkbox next to your project name, and **Install**.
3. Replace `Program.cs` with the following code.
4. Replace `<Subscription Key>` with your valid subscription key.
5. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
6. Run the program.
7. At the prompt, enter the path to a local image.

The thumbnail is saved to the same folder as the local image, using the original name with the suffix "_thumb".

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Replace <Subscription Key> with your valid subscription key.
        const string subscriptionKey = "<Subscription Key>";

        // You must use the same region in your REST call as you used to
        // get your subscription keys. For example, if you got your
        // subscription keys from westus, replace "westcentralus" in the URL
        // below with "westus".
        //
        // Free trial subscription keys are generated in the westcentralus region.
        // If you use a free trial subscription key, you shouldn't need to change
        // this region.
        const string uriBase =
            "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/generateThumbnail";
```

```

static void Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Thumbnail:");
    Console.Write(
        "Enter the path to the image you wish to use to create a thumbnail image: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Make the REST API call.
        Console.WriteLine("\nWait a moment for the results to appear.\n");
        MakeThumbNailRequest(imageFilePath).Wait();
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets a thumbnail image from the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file to use to create the thumbnail image.</param>
static async Task MakeThumbNailRequest(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameters.
        string requestParameters = "width=200&height=150&smartCropping=true";

        // Assemble the URI for the REST API Call.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Request body.
        // Posts a locally stored JPEG image.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses content type "application/octet-stream".
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");

            // Make the REST API call.
            response = await client.PostAsync(uri, content);
        }

        if (response.IsSuccessStatusCode)
        {
            // Display the response data.
            Console.WriteLine("\nResponse:\n{0}", response);

            // Get the image data.

```

```

        byte[] thumbnailImageData =
            await response.Content.ReadAsByteArrayAsync();

        // Save the thumbnail to the same folder as the original image,
        // using the original name with the suffix "_thumb".
        // Note: This will overwrite an existing file of the same name.
        string thumbnailFilePath =
            imageFilePath.Insert(imageFilePath.Length - 4, "_thumb");
        File.WriteAllBytes(thumbnailFilePath, thumbnailImageData);
        Console.WriteLine("\nThumbnail written to: {0}", thumbnailFilePath);
    }
    else
    {
        // Display the JSON error data.
        string errorString = await response.Content.ReadAsStringAsync();
        Console.WriteLine("\n\nResponse:\n{0}\n",
            JToken.Parse(errorString).ToString());
    }
}
catch (Exception e)
{
    Console.WriteLine("\n" + e.Message);
}
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

Get Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong.

Response:

```

StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.StreamContent, Headers:
{
    Pragma: no-cache
    apim-request-id: 131eb5b4-5807-466d-9656-4c1ef0a64c9b
    Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
    x-content-type-options: nosniff
    Cache-Control: no-cache
    Date: Tue, 06 Jun 2017 20:54:07 GMT
    X-AspNet-Version: 4.0.30319
    X-Powered-By: ASP.NET
    Content-Length: 5800
    Content-Type: image/jpeg
    Expires: -1
}

```

Next steps

Explore a basic Windows application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API C# Tutorial](#)

Quickstart: Extract printed text (OCR) with C#

6/15/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Create a new Visual C# Console App in Visual Studio.
2. Install the Newtonsoft.Json NuGet package.
 - a. On the menu, click **Tools**, select **NuGet Package Manager**, then **Manage NuGet Packages for Solution**.
 - b. Click the **Browse** tab, and in the **Search** box type "Newtonsoft.Json".
 - c. Select **Newtonsoft.Json** when it displays, then click the checkbox next to your project name, and **Install**.
3. Replace `Program.cs` with the following code.
4. Replace `<Subscription Key>` with your valid subscription key.
5. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
6. Run the program.
7. At the prompt, enter the path to a local image.

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Replace <Subscription Key> with your valid subscription key.
        const string subscriptionKey = "<Subscription Key>";

        // You must use the same region in your REST call as you used to
        // get your subscription keys. For example, if you got your
        // subscription keys from westus, replace "westcentralus" in the URL
        // below with "westus".
        //
        // Free trial subscription keys are generated in the westcentralus region.
        // If you use a free trial subscription key, you shouldn't need to change
        // this region.
        const string uriBase =
            "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/ocr";

        static void Main()
```

```

static void Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Optical Character Recognition:");
    Console.Write("Enter the path to an image with text you wish to read: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Make the REST API call.
        Console.WriteLine("\nWait a moment for the results to appear.\n");
        MakeOCRRequest(imageFilePath).Wait();
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets the text visible in the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file with printed text.</param>
static async Task MakeOCRRequest(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameters.
        string requestParameters = "language=unk&detectOrientation=true";

        // Assemble the URI for the REST API Call.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Request body. Posts a locally stored JPEG image.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses content type "application/octet-stream".
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");

            // Make the REST API call.
            response = await client.PostAsync(uri, content);
        }

        // Get the JSON response.
        string contentString = await response.Content.ReadAsStringAsync();

        // Display the JSON response.
        Console.WriteLine("\nResponse:\n\n{0}\n",
            JToken.Parse(contentString).ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("\n" + e.Message);
    }
}

```

```

    }
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

OCR response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words, for example:

```

{
  "language": "en",
  "textAngle": -1.5000000000000335,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "154,49,351,575",
      "lines": [
        {
          "boundingBox": "165,49,340,117",
          "words": [
            {
              "boundingBox": "165,49,63,109",
              "text": "A"
            },
            {
              "boundingBox": "261,50,244,116",
              "text": "GOAL"
            }
          ]
        }
      ],
    },
    {
      "boundingBox": "165,169,339,93",
      "words": [
        {
          "boundingBox": "165,169,339,93",
          "text": "WITHOUT"
        }
      ]
    },
    {
      "boundingBox": "159,264,342,117",
      "words": [
        {
          "boundingBox": "159,264,64,110",
          "text": "A"
        },
        {
          "boundingBox": "255,266,246,115",
          "text": "PLAN"
        }
      ]
    }
  ],
  ,

```

```

    },
    {
      "boundingBox": "161,384,338,119",
      "words": [
        {
          "boundingBox": "161,384,86,113",
          "text": "IS"
        },
        {
          "boundingBox": "274,387,225,116",
          "text": "JUST"
        }
      ]
    },
  ],
  {
    "boundingBox": "154,506,341,118",
    "words": [
      {
        "boundingBox": "154,506,62,111",
        "text": "A"
      },
      {
        "boundingBox": "248,508,247,116",
        "text": "WISH"
      }
    ]
  }
]
}

```

Next steps

Explore a basic Windows application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API C# Tutorial](#)

Quickstart: Extract handwritten text with C#

6/15/2018 • 5 minutes to read • [Edit Online](#)

In this quickstart, you extract handwritten text from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Recognize Text request

With the [Recognize Text](#) and the [Get Recognize Text Operation Result methods](#), you can detect handwritten text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Create a new Visual C# Console App in Visual Studio.
2. Install the Newtonsoft.Json NuGet package.
 - a. On the menu, click **Tools**, select **NuGet Package Manager**, then **Manage NuGet Packages for Solution**.
 - b. Click the **Browse** tab, and in the **Search** box type "Newtonsoft.Json".
 - c. Select **Newtonsoft.Json** when it displays, then click the checkbox next to your project name, and **Install**.
3. Replace `Program.cs` with the following code.
4. Replace `<Subscription Key>` with your valid subscription key.
5. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
6. Run the program.
7. At the prompt, enter the path to a local image.

```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace CSHttpClientSample
{
    static class Program
    {
        // Replace <Subscription Key> with your valid subscription key.
        const string subscriptionKey = "<Subscription Key>";

        // You must use the same region in your REST call as you used to
        // get your subscription keys. For example, if you got your
        // subscription keys from westus, replace "westcentralus" in the URL
        // below with "westus".
        //
        // Free trial subscription keys are generated in the westcentralus region.
        // If you use a free trial subscription key, you shouldn't need to change
        // this region.
        const string uriBase =
            "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/recognizeText";

        static void Main()
```

```

{
    // Get the path and filename to process from the user.
    Console.WriteLine("Handwriting Recognition:");
    Console.Write(
        "Enter the path to an image with handwritten text you wish to read: ");
    string imageFilePath = Console.ReadLine();

    if (File.Exists(imageFilePath))
    {
        // Make the REST API call.
        Console.WriteLine("\nWait a moment for the results to appear.\n");
        ReadHandwrittenText(imageFilePath).Wait();
    }
    else
    {
        Console.WriteLine("\nInvalid file path");
    }
    Console.WriteLine("\nPress Enter to exit...");
    Console.ReadLine();
}

/// <summary>
/// Gets the handwritten text from the specified image file by using
/// the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file with handwritten text.</param>
static async Task ReadHandwrittenText(string imageFilePath)
{
    try
    {
        HttpClient client = new HttpClient();

        // Request headers.
        client.DefaultRequestHeaders.Add(
            "Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request parameter.
        // Note: The request parameter changed for APIv2.
        // For APIv1, it is "handwriting=true".
        string requestParameters = "mode=Handwritten";

        // Assemble the URI for the REST API Call.
        string uri = uriBase + "?" + requestParameters;

        HttpResponseMessage response;

        // Two REST API calls are required to extract handwritten text.
        // One call to submit the image for processing, the other call
        // to retrieve the text found in the image.
        // operationLocation stores the REST API location to call to
        // retrieve the text.
        string operationLocation;

        // Request body.
        // Posts a locally stored JPEG image.
        byte[] byteData = GetImageAsByteArray(imageFilePath);

        using (ByteArrayContent content = new ByteArrayContent(byteData))
        {
            // This example uses content type "application/octet-stream".
            // The other content types you can use are "application/json"
            // and "multipart/form-data".
            content.Headers.ContentType =
                new MediaTypeHeaderValue("application/octet-stream");

            // The first REST call starts the async process to analyze the
            // written text in the image.
            response = await client.PostAsync(uri, content);
        }
    }
}

```

```

// The response contains the URI to retrieve the result of the process.
if (response.IsSuccessStatusCode)
    operationLocation =
        response.Headers.GetValues("Operation-Location").FirstOrDefault();
else
{
    // Display the JSON error data.
    string errorString = await response.Content.ReadAsStringAsync();
    Console.WriteLine("\n\nResponse:\n{0}\n",
        JToken.Parse(errorString).ToString());
    return;
}

// The second REST call retrieves the text written in the image.
//
// Note: The response may not be immediately available. Handwriting
// recognition is an async operation that can take a variable amount
// of time depending on the length of the handwritten text. You may
// need to wait or retry this operation.
//
// This example checks once per second for ten seconds.
string contentString;
int i = 0;
do
{
    System.Threading.Thread.Sleep(1000);
    response = await client.GetAsync(operationLocation);
    contentString = await response.Content.ReadAsStringAsync();
    ++i;
}
while (i < 10 && contentString.IndexOf("\"status\":\"Succeeded\"") == -1);

if (i == 10 && contentString.IndexOf("\"status\":\"Succeeded\"") == -1)
{
    Console.WriteLine("\nTimeout error.\n");
    return;
}

// Display the JSON response.
Console.WriteLine("\nResponse:\n\n{0}\n",
    JToken.Parse(contentString).ToString());
}
catch (Exception e)
{
    Console.WriteLine("\n" + e.Message);
}
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    using (FileStream fileStream =
        new FileStream(imageFilePath, FileMode.Open, FileAccess.Read))
    {
        BinaryReader binaryReader = new BinaryReader(fileStream);
        return binaryReader.ReadBytes((int)fileStream.Length);
    }
}
}
}

```

Recognize Text response

A successful response is returned in JSON, for example:

```
{
  "status": "Succeeded",
  "recognitionResult": {
    "lines": [
      {
        "boundingBox": [
          99,
          195,
          1309,
          45,
          1340,
          292,
          130,
          442
        ],
        "text": "when you write them down",
        "words": [
          {
            "boundingBox": [
              152,
              191,
              383,
              154,
              341,
              421,
              110,
              458
            ],
            "text": "when"
          },
          {
            "boundingBox": [
              436,
              145,
              607,
              118,
              565,
              385,
              394,
              412
            ],
            "text": "you"
          },
          {
            "boundingBox": [
              644,
              112,
              873,
              76,
              831,
              343,
              602,
              379
            ],
            "text": "write"
          }
        ]
      },
      {
        "boundingBox": [
          895,
          72,
          1092,
          41,
          1050,
          ...
        ]
      }
    ]
  }
}
```



```

        308,
        853,
        339
    ],
    "text": "them"
},
{
    "boundingBox": [
        1140,
        33,
        1400,
        0,
        1359,
        258,
        1098,
        300
    ],
    "text": "down"
}
]
},
{
    "boundingBox": [
        142,
        222,
        1252,
        62,
        1269,
        180,
        159,
        340
    ],
    "text": "You remember things better",
    "words": [
        {
            "boundingBox": [
                140,
                223,
                267,
                205,
                288,
                324,
                162,
                342
            ],
            "text": "You"
        },
        {
            "boundingBox": [
                314,
                198,
                740,
                137,
                761,
                256,
                335,
                317
            ],
            "text": "remember"
        }
    ],
    {
        "boundingBox": [
            761,
            134,
            1026,
            95,
            1047,
            215,
            782,

```

```

        253
    ],
    "text": "things"
  },
  {
    "boundingBox": [
      1046,
      92,
      1285,
      58,
      1307,
      177,
      1068,
      212
    ],
    "text": "better"
  }
]
},
{
  "boundingBox": [
    155,
    405,
    537,
    338,
    557,
    449,
    175,
    516
  ],
  "text": "by hand",
  "words": [
    {
      "boundingBox": [
        146,
        408,
        266,
        387,
        301,
        495,
        181,
        516
      ],
      "text": "by"
    },
    {
      "boundingBox": [
        290,
        383,
        569,
        334,
        604,
        443,
        325,
        491
      ],
      "text": "hand"
    }
  ]
}
]
}
}
}

```

Next steps

Explore a basic Windows application that uses Computer Vision to perform optical character recognition (OCR);

create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API C# Tutorial](#)

Quickstart: Analyze a remote image with cURL

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you analyze a remote image to extract visual features using Computer Vision. To analyze a local image, see [Analyze a local image with cURL](#).

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze a remote image

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the Request URL (`https://westcentralus.api.cognitive.microsoft.com/vision/v2.0`) to use the location where you obtained your subscription keys, if necessary.
4. Optionally, change the image (`{\"ur1\": \"...\"}`) to analyze.
5. Optionally, change the response language (`language=en`).
6. Open a command window on a computer with cURL installed.
7. Paste the code in the window and run the command.

NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

Analyze Image request

```
curl -H "Ocp-Apim-Subscription-Key: <Subscription Key>" -H "Content-Type: application/json"
"https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze?
visualFeatures=Categories,Description&details=Landmarks&language=en" -d "
{\"ur1\": \"http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg\"}"
```

Analyze Image response

A successful response is returned in JSON, for example:

```
{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "requestId": "b6e33879-abb2-43a0-a96e-02cb5ae0b795",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Analyze a local image with cURL

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you analyze a local image to extract visual features using Computer Vision. To analyze a remote image, see [Analyze a remote image with cURL](#).

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze a local image

This sample is similar to [Analyze a remote image with cURL](#) except the image to analyze is read locally from disk. Three changes are required:

- Change the Content-Type to `"Content-Type: application/octet-stream"`.
- Change the `-d` switch to `--data-binary`.
- Specify the image to analyze using the following syntax: `@C:/Pictures/ImageToAnalyze.jpg`.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the Request URL (`https://westcentralus.api.cognitive.microsoft.com/vision/v2.0`) to use the location where you obtained your subscription keys, if necessary.
4. Replace `<Image To Analyze>` with the local image you want to analyze.
5. Optionally, change the response language (`language=en`).
6. Open a command window on a computer with cURL installed.
7. Paste the code in the window and run the command.

NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

```
curl -H "Ocp-Apim-Subscription-Key: <Subscription Key>" -H "Content-Type: application/octet-stream"
"https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze?
visualFeatures=Categories,Description&details=Landmarks&language=en" --data-binary <Image To Analyze>
```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "requestId": "b6e33879-abb2-43a0-a96e-02cb5ae0b795",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}

```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Generate a thumbnail with cURL

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Replace `<File>` with the path and filename to save the thumbnail.
4. Change the Request URL (`https://westcentralus.api.cognitive.microsoft.com/vision/v2.0`) to use the location where you obtained your subscription keys, if necessary.
5. Optionally, change the image (`{\"url\": \"...\"}`) to analyze.
6. Open a command window on a computer with cURL installed.
7. Paste the code in the window and run the command.

NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

```
curl -H "Ocp-Apim-Subscription-Key: <Subscription Key>" -o <File> -H "Content-Type: application/json"
"https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/generateThumbnail?
width=100&height=100&smartCropping=true" -d "
{\"url\": \"https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Shorkie_Poo_Puppy.jpg/1280px-
Shorkie_Poo_Puppy.jpg\"}"
```

Get Thumbnail response

A successful response writes the thumbnail image to `<File>`. If the request fails, the response contains an error code and a message to help determine what went wrong.

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Extract printed text (OCR) with cURL

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the Request URL (`https://westcentralus.api.cognitive.microsoft.com/vision/v2.0`) to use the location where you obtained your subscription keys, if necessary.
4. Optionally, change the image (`{\"url\":\"...\"}`) to analyze.
5. Open a command window on a computer with cURL installed.
6. Paste the code in the window and run the command.

NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

```
curl -H "Ocp-Apim-Subscription-Key: <Subscription Key>" -H "Content-Type: application/json"
"https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/ocr?language=unk&detectOrientation=true" -d "
{\"url\":\"https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-
Atomist_quote_from_Democritus.png\"}"
```

OCR response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words, for example:

```
{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "text": "Atomist Quote",
              "boundingBox": "28,16,288,41"
            }
          ]
        }
      ]
    }
  ]
}
```

```

        "boundingBox": "28,16,288,41",
        "text": "NOTHING"
    }
]
},
{
    "boundingBox": "27,66,283,52",
    "words": [
        {
            "boundingBox": "27,66,283,52",
            "text": "EXISTS"
        }
    ]
},
{
    "boundingBox": "27,128,292,49",
    "words": [
        {
            "boundingBox": "27,128,292,49",
            "text": "EXCEPT"
        }
    ]
},
{
    "boundingBox": "24,188,292,54",
    "words": [
        {
            "boundingBox": "24,188,292,54",
            "text": "ATOMS"
        }
    ]
},
{
    "boundingBox": "22,253,297,32",
    "words": [
        {
            "boundingBox": "22,253,105,32",
            "text": "AND"
        },
        {
            "boundingBox": "144,253,175,32",
            "text": "EMPTY"
        }
    ]
},
{
    "boundingBox": "21,298,304,60",
    "words": [
        {
            "boundingBox": "21,298,304,60",
            "text": "SPACE."
        }
    ]
},
{
    "boundingBox": "26,387,294,37",
    "words": [
        {
            "boundingBox": "26,387,210,37",
            "text": "Everything"
        },
        {
            "boundingBox": "249,389,71,27",
            "text": "else"
        }
    ]
},
{
    "boundingBox": "127,431,198,36",

```

```
    "words": [  
      {  
        "boundingBox": "127,431,31,29",  
        "text": "is"  
      },  
      {  
        "boundingBox": "172,431,153,36",  
        "text": "opinion."  
      }  
    ]  
  }  
]  
}  
]
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Analyze an image with Java

6/18/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you analyze an image to extract visual features using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze Image request

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Create a new command-line app.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace `<Subscription Key>` with your valid subscription key.
4. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
5. Optionally, change the `imageToAnalyze` value to another image.
6. Download these libraries from the Maven Repository to the `lib` directory in your project:

- `org.apache.httpcomponents:httpclient:4.5.5`
- `org.apache.httpcomponents:httpcore:4.4.9`
- `org.json:json:20180130`

7. Run 'Main'.

```
// This sample uses the following libraries:
// - Apache HTTP client (org.apache.httpcomponents:httpclient:4.5.5)
// - Apache HTTP core (org.apache.httpcomponents:httpcore:4.4.9)
// - JSON library (org.json:json:20180130).

import java.net.URI;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;
```

```

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    private static final String subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
    // subscription keys. For example, if you got your subscription keys from
    // westus, replace "westcentralus" in the URI below with "westus".
    //
    // Free trial subscription keys are generated in the westcentralus region. If you
    // use a free trial subscription key, you shouldn't need to change this region.
    private static final String uriBase =
        "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze";

    private static final String imageToAnalyze =
        "https://upload.wikimedia.org/wikipedia/commons/" +
        "1/12/Broadway_and_Times_Square_by_night.jpg";

    public static void main(String[] args) {
        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            URIBuilder builder = new URIBuilder(uriBase);

            // Request parameters. All of them are optional.
            builder.setParameter("visualFeatures", "Categories,Description,Color");
            builder.setParameter("language", "en");

            // Prepare the URI for the REST API call.
            URI uri = builder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body.
            StringEntity requestEntity =
                new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
            request.setEntity(requestEntity);

            // Make the REST API call and get the response entity.
            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null) {
                // Format and display the JSON response.
                String jsonString = EntityUtils.toString(entity);
                JSONObject json = new JSONObject(jsonString);
                System.out.println("REST Response:\n");
                System.out.println(json.toString(2));
            }
        } catch (Exception e) {
            // Display error message.
            System.out.println(e.getMessage());
        }
    }
}

```

Analyze Image response

A successful response is returned in JSON, for example:

REST Response:

```
{
  "metadata": {
    "width": 1826,
    "format": "Jpeg",
    "height": 2436
  },
  "color": {
    "dominantColorForeground": "Brown",
    "isBWImg": false,
    "accentColor": "B74314",
    "dominantColorBackground": "Brown",
    "dominantColors": ["Brown"]
  },
  "requestId": "bbffe1a1-4fa3-4a6b-a4d5-a4964c58a811",
  "description": {
    "captions": [{
      "confidence": 0.8241405091548035,
      "text": "a group of people on a city street filled with traffic at night"
    }],
    "tags": [
      "outdoor",
      "building",
      "street",
      "city",
      "busy",
      "people",
      "filled",
      "traffic",
      "many",
      "table",
      "car",
      "group",
      "walking",
      "bunch",
      "crowded",
      "large",
      "night",
      "light",
      "standing",
      "man",
      "tall",
      "umbrella",
      "riding",
      "sign",
      "crowd"
    ]
  },
  "categories": [{
    "score": 0.625,
    "name": "outdoor_street"
  }]
}
```

Next steps

Explore a Java Swing application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Java Tutorial](#)

Quickstart: Generate a thumbnail with Java

6/15/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Create a new command-line app.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace `<Subscription Key>` with your valid subscription key.
4. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
5. Optionally, change the `imageToAnalyze` value to another image.
6. Download these libraries from the Maven Repository to the `lib` directory in your project:
 - `org.apache.httpcomponents:httpclient:4.5.5`
 - `org.apache.httpcomponents:httpcore:4.4.9`
 - `org.json:json:20180130`
7. Run 'Main'.

```
// This sample uses the following libraries:
// - Apache HTTP client (org.apache.httpcomponents:httpclient:4.5.5)
// - Apache HTTP core (org.apache.httpcomponents:httpcore:4.4.9)
// - JSON library (org.json:json:20180130).

import java.awt.*;
import javax.swing.*;
import java.net.URI;
import java.io.InputStream;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****
```

```

// Replace <Subscription Key> with your valid subscription key.
private static final String subscriptionKey = "<Subscription Key>";

// You must use the same region in your REST call as you used to get your
// subscription keys. For example, if you got your subscription keys from
// westus, replace "westcentralus" in the URI below with "westus".
//
// Free trial subscription keys are generated in the westcentralus region. If you
// use a free trial subscription key, you shouldn't need to change this region.
private static final String uriBase =
    "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/generateThumbnail";

private static final String imageToAnalyze =
    "https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg";

public static void main(String[] args) {
    CloseableHttpClient httpClient = HttpClientBuilder.create().build();

    try {
        URIBuilder uriBuilder = new URIBuilder(uriBase);

        // Request parameters.
        uriBuilder.setParameter("width", "100");
        uriBuilder.setParameter("height", "150");
        uriBuilder.setParameter("smartCropping", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request body.
        StringEntity requestEntity =
            new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
        request.setEntity(requestEntity);

        // Make the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // Check for success.
        if (response.getStatusLine().getStatusCode() == 200) {
            // Display the thumbnail.
            System.out.println("\nDisplaying thumbnail.\n");
            displayImage(entity.getContent());
        } else {
            // Format and display the JSON error message.
            String jsonString = EntityUtils.toString(entity);
            JSONObject json = new JSONObject(jsonString);
            System.out.println("Error:\n");
            System.out.println(json.toString(2));
        }
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

// Displays the given input stream as an image.
private static void displayImage(InputStream inputStream) {
    try {
        BufferedImage bufferedImage = ImageIO.read(inputStream);

        ImageIcon imageIcon = new ImageIcon(bufferedImage);

        JLabel jLabel = new JLabel();
        jLabel.setIcon(imageIcon);
    }
}

```



```

        JLabel.setIcon(imageIcon);

        JFrame jFrame = new JFrame();
        jFrame.setLayout(new FlowLayout());
        jFrame.setSize(100, 150);

        jFrame.add(jLabel);
        jFrame.setVisible(true);
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
}

```

Get Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong. The following text is an example of a successful response.

Response:

```

StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.StreamContent, Headers:
{
  Pragma: no-cache
  apim-request-id: 131eb5b4-5807-466d-9656-4c1ef0a64c9b
  Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
  x-content-type-options: nosniff
  Cache-Control: no-cache
  Date: Tue, 06 Jun 2017 20:54:07 GMT
  X-AspNet-Version: 4.0.30319
  X-Powered-By: ASP.NET
  Content-Length: 5800
  Content-Type: image/jpeg
  Expires: -1
}

```

Next steps

Explore a Java Swing application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Java Tutorial](#)

Quickstart: Extract printed text (OCR) with Java

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Create a new command-line app.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace `<Subscription Key>` with your valid subscription key.
4. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
5. Optionally, change the `imageToAnalyze` value to another image.
6. Download these libraries from the Maven Repository to the `lib` directory in your project:
 - `org.apache.httpcomponents:httpclient:4.5.5`
 - `org.apache.httpcomponents:httpcore:4.4.9`
 - `org.json:json:20180130`
7. Run 'Main'.

```
// This sample uses the following libraries:
// - Apache HTTP client (org.apache.httpcomponents:httpclient:4.5.5)
// - Apache HTTP core (org.apache.httpcomponents:httpcore:4.4.9)
// - JSON library (org.json:json:20180130).

import java.net.URI;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    private static final String subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
    // subscription keys. For example, if you got your subscription keys from
```

```

// Subscription key. For example, if you get your subscription key from
// westus, replace "westcentralus" in the URI below with "westus".
//
// Free trial subscription keys are generated in the westcentralus region. If you
// use a free trial subscription key, you shouldn't need to change this region.
private static final String uriBase =
    "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/ocr";

private static final String imageToAnalyze =
    "https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/" +
    "Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png";

public static void main(String[] args) {
    CloseableHttpClient httpClient = HttpClientBuilder.create().build();

    try {
        URIBuilder uriBuilder = new URIBuilder(uriBase);

        uriBuilder.setParameter("language", "unk");
        uriBuilder.setParameter("detectOrientation", "true");

        // Request parameters.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request body.
        StringEntity requestEntity =
            new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
        request.setEntity(requestEntity);

        // Make the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        if (entity != null) {
            // Format and display the JSON response.
            String jsonString = EntityUtils.toString(entity);
            JSONObject json = new JSONObject(jsonString);
            System.out.println("REST Response:\n");
            System.out.println(json.toString(2));
        }
    } catch (Exception e) {
        // Display error message.
        System.out.println(e.getMessage());
    }
}

```

OCR response

A successful response is returned in JSON. The OCR results include the detected text and bounding boxes for regions, lines, and words.

The program should produce output similar to the following JSON:

```

REST Response:

{
  "orientation": "Up",
  "regions": [{
    "boundingBox": "21,16,304,451",
    "lines": [

```

```

{
  "boundingBox": "28,16,288,41",
  "words": [{
    "boundingBox": "28,16,288,41",
    "text": "NOTHING"
  }]
},
{
  "boundingBox": "27,66,283,52",
  "words": [{
    "boundingBox": "27,66,283,52",
    "text": "EXISTS"
  }]
},
{
  "boundingBox": "27,128,292,49",
  "words": [{
    "boundingBox": "27,128,292,49",
    "text": "EXCEPT"
  }]
},
{
  "boundingBox": "24,188,292,54",
  "words": [{
    "boundingBox": "24,188,292,54",
    "text": "ATOMS"
  }]
},
{
  "boundingBox": "22,253,297,32",
  "words": [
    {
      "boundingBox": "22,253,105,32",
      "text": "AND"
    },
    {
      "boundingBox": "144,253,175,32",
      "text": "EMPTY"
    }
  ]
},
{
  "boundingBox": "21,298,304,60",
  "words": [{
    "boundingBox": "21,298,304,60",
    "text": "SPACE."
  }]
},
{
  "boundingBox": "26,387,294,37",
  "words": [
    {
      "boundingBox": "26,387,210,37",
      "text": "Everything"
    },
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ]
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {

```

```
        "boundingBox": "172,431,153,36",  
        "text": "opinion."  
    }  
]  
}  
],  
"textAngle": 0,  
"language": "en"  
}
```

Next steps

Explore a Java Swing application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Java Tutorial](#)

Quickstart: Extract handwritten text with Java

6/15/2018 • 5 minutes to read • [Edit Online](#)

In this quickstart, you extract handwritten txt from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Recognize Text request

With the [Recognize Text](#) and the [Get Recognize Text Operation Result methods](#), you can detect handwritten text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Create a new command-line app.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace `<Subscription Key>` with your valid subscription key.
4. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
5. Optionally, change the `imageToAnalyze` value to another image.
6. Download these libraries from the Maven Repository to the `lib` directory in your project:
 - `org.apache.httpcomponents:httpclient:4.5.5`
 - `org.apache.httpcomponents:httpcore:4.4.9`
 - `org.json:json:20180130`
7. Run 'Main'.

```
// This sample uses the following libraries:
// - Apache HTTP client (org.apache.httpcomponents:httpclient:4.5.5)
// - Apache HTTP core (org.apache.httpcomponents:httpcore:4.4.9)
// - JSON library (org.json:json:20180130).

import java.net.URI;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import org.apache.http.Header;
import org.json.JSONObject;

public class Main {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    private static final String subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
```

```

// subscription keys. For example, if you got your subscription keys from
// westus, replace "westcentralus" in the URI below with "westus".
//
// Free trial subscription keys are generated in the westcentralus region. If you
// use a free trial subscription key, you shouldn't need to change this region.
private static final String uriBase =
    "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/recognizeText";

private static final String imageToAnalyze =
    "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/" +
    "Cursive_Writing_on_Notebook_paper.jpg/800px-Cursive_Writing_on_Notebook_paper.jpg";

public static void main(String[] args) {
    CloseableHttpClient httpTextClient = HttpClientBuilder.create().build();
    CloseableHttpClient httpResultClient = HttpClientBuilder.create().build();

    try {
        // This operation requires two REST API calls. One to submit the image
        // for processing, the other to retrieve the text found in the image.

        URIBuilder builder = new URIBuilder(uriBase);

        // Request parameter.
        // Note: The request parameter changed for APIv2.
        // For APIv1, it is "handwriting", "true".
        builder.setParameter("mode", "Handwritten");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

        // Request body.
        StringEntity requestEntity =
            new StringEntity("{\"url\":\"" + imageToAnalyze + "\"}");
        request.setEntity(requestEntity);

        // Make the first REST API call to detect the text.
        HttpResponse response = httpTextClient.execute(request);

        // Check for success.
        if (response.getStatusLine().getStatusCode() != 202) {
            // Format and display the JSON error message.
            HttpEntity entity = response.getEntity();
            String jsonString = EntityUtils.toString(entity);
            JSONObject json = new JSONObject(jsonString);
            System.out.println("Error:\n");
            System.out.println(json.toString(2));
            return;
        }

        // Stores the URI where you can get the text recognition operation result.
        String operationLocation = null;

        // 'Operation-Location' in the response contains the URI to
        // retrieve the recognized text.
        Header[] responseHeaders = response.getAllHeaders();
        for (Header header : responseHeaders) {
            if (header.getName().equals("Operation-Location")) {
                operationLocation = header.getValue();
                break;
            }
        }

        if (operationLocation == null) {
            System.out.println("\nError retrieving Operation-Location.\nExiting.");
        }
    }
}

```

```

        System.exit(1);
    }

    // Note: The response may not be immediately available. Handwriting
    // recognition is an asynchronous operation, which takes a variable
    // amount of time dependent on the length of the text analyzed. You
    // may need to wait or retry the Get operation.

    System.out.println("\nHandwritten text submitted.\n" +
        "Waiting 10 seconds to retrieve the recognized text.\n");
    Thread.sleep(10000);

    // Make the second REST API call and get the response.
    HttpGet resultRequest = new HttpGet(operationLocation);
    resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

    HttpResponseMessage resultResponse = httpResultClient.execute(resultRequest);
    HttpEntity responseEntity = resultResponse.getEntity();

    if (responseEntity != null) {
        // Format and display the JSON response.
        String jsonString = EntityUtils.toString(responseEntity);
        JSONObject json = new JSONObject(jsonString);
        System.out.println("Text recognition result response: \n");
        System.out.println(json.toString(2));
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}
}

```

Recognize Text response

A successful response is returned in JSON. The handwriting recognition results include the detected text and bounding boxes for regions, lines, and words.

The program should produce output similar to the following JSON:

```
Handwritten text submitted. Waiting 10 seconds to retrieve the recognized text.
```

```
Text recognition result response:
```

```
{
  "status": "Succeeded",
  "recognitionResult": {"lines": [
    {
      "boundingBox": [
        2,
        84,
        783,
        96,
        782,
        154,
        1,
        148
      ],
      "words": [
        {
          "boundingBox": [
            6,
            86,
            92,
            87,
            71,
            154
          ]
        }
      ]
    }
  ]
}
```



```
151,  
0,  
150  
],  
"text": "Pack"  
},  
{  
  "boundingBox": [  
    86,  
    87,  
    172,  
    88,  
    150,  
    152,  
    64,  
    151  
  ],  
  "text": "my"  
},  
{  
  "boundingBox": [  
    165,  
    88,  
    241,  
    89,  
    219,  
    152,  
    144,  
    152  
  ],  
  "text": "box"  
},  
{  
  "boundingBox": [  
    234,  
    89,  
    343,  
    90,  
    322,  
    154,  
    213,  
    152  
  ],  
  "text": "with"  
},  
{  
  "boundingBox": [  
    347,  
    90,  
    432,  
    91,  
    411,  
    154,  
    325,  
    154  
  ],  
  "text": "five"  
},  
{  
  "boundingBox": [  
    432,  
    91,  
    538,  
    92,  
    516,  
    154,  
    411,  
    154  
  ],  
  "text": "with"
```

```

      "text": "dozen"
    },
    {
      "boundingBox": [
        554,
        92,
        696,
        94,
        675,
        154,
        533,
        154
      ],
      "text": "liquor"
    },
    {
      "boundingBox": [
        710,
        94,
        800,
        96,
        800,
        154,
        688,
        154
      ],
      "text": "jugs"
    }
  ],
  "text": "Pack my box with five dozen liquor jugs"
},
{
  "boundingBox": [
    2,
    52,
    65,
    46,
    69,
    89,
    7,
    95
  ],
  "words": [{
    "boundingBox": [
      0,
      62,
      79,
      39,
      94,
      82,
      0,
      105
    ],
    "text": "dog"
  }],
  "text": "dog"
},
{
  "boundingBox": [
    6,
    2,
    771,
    13,
    770,
    75,
    5,
    64
  ],
  "words": [

```

```
{
  "boundingBox": [
    8,
    4,
    92,
    5,
    77,
    71,
    0,
    71
  ],
  "text": "The"
},
{
  "boundingBox": [
    89,
    5,
    188,
    5,
    173,
    72,
    74,
    71
  ],
  "text": "quick"
},
{
  "boundingBox": [
    188,
    5,
    323,
    6,
    308,
    73,
    173,
    72
  ],
  "text": "brown"
},
{
  "boundingBox": [
    316,
    6,
    386,
    6,
    371,
    73,
    302,
    73
  ],
  "text": "fox"
},
{
  "boundingBox": [
    396,
    7,
    508,
    7,
    493,
    74,
    381,
    73
  ],
  "text": "jumps"
},
{
  "boundingBox": [
    501,
    7,
```

```

        604,
        8,
        589,
        75,
        487,
        74
    ],
    "text": "over"
},
{
    "boundingBox": [
        600,
        8,
        673,
        8,
        658,
        75,
        586,
        75
    ],
    "text": "the"
},
{
    "boundingBox": [
        670,
        8,
        800,
        9,
        787,
        76,
        655,
        75
    ],
    "text": "lazy"
}
],
"text": "The quick brown fox jumps over the lazy"
}
]}
}

```

Next steps

Explore a Java Swing application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Java Tutorial](#)

Quickstart: Analyze an image with JavaScript

6/18/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you analyze an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze Image request

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Copy the following and save it to a file such as `analyze.html`.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
4. Drag-and-drop the file into your browser.
5. Click the `Analyze image` button.

This sample uses jQuery 1.9.0. For a sample that uses JavaScript without jQuery, see [Intelligently generate a thumbnail](#).

```
<!DOCTYPE html>
<html>
<head>
  <title>Analyze Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    var subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
    // subscription keys. For example, if you got your subscription keys from
    // westus, replace "westcentralus" in the URI below with "westus".
    //
```

```

// Free trial subscription keys are generated in the westcentralus region.
// If you use a free trial subscription key, you shouldn't need to change
// this region.
var uriBase =
    "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze";

// Request parameters.
var params = {
    "visualFeatures": "Categories,Description,Color",
    "details": "",
    "language": "en",
};

// Display the image.
var sourceImageUrl = document.getElementById("inputImage").value;
document.querySelector("#sourceImage").src = sourceImageUrl;

// Make the REST API call.
$.ajax({
    url: uriBase + "?" + $.param(params),

    // Request headers.
    beforeSend: function(xhrObj){
        xhrObj.setRequestHeader("Content-Type","application/json");
        xhrObj.setRequestHeader(
            "Ocp-Apim-Subscription-Key", subscriptionKey);
    },

    type: "POST",

    // Request body.
    data: '{"url": ' + "'" + sourceImageUrl + "'",
})

.done(function(data) {
    // Show formatted JSON on webpage.
    $("#responseTextArea").val(JSON.stringify(data, null, 2));
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " :
        errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" :
        jQuery.parseJSON(jqXHR.responseText).message;
    alert(errorString);
});
};
</script>

<h1>Analyze image:</h1>
Enter the URL to an image, then click the <strong>Analyze image</strong> button.
<br><br>
Image to analyze:
<input type="text" name="inputImage" id="inputImage"
    value="http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg" />
<button onclick="processImage()">Analyze image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput"
            style="width:580px; height:400px;"></textarea>
    </div>
    <div id="imageDiv" style="width:420px; display:table-cell;">
        Source image:
        <br><br>
        <img id="sourceImage" width="400" />
    </div>
</div>

```

```
</div>  
</div>  
</body>  
</html>
```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "color": {
    "dominantColorForeground": "Grey",
    "dominantColorBackground": "Green",
    "dominantColors": [
      "Grey",
      "Green"
    ],
    "accentColor": "4D5E2F",
    "isBwImg": false
  },
  "requestId": "73ef10ce-a4ea-43c6-ae7-70325777e4b3",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}

```

Next steps

Explore a JavaScript application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API JavaScript Tutorial](#)

Quickstart: Generate a thumbnail with JavaScript

6/15/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Copy the following and save it to a file such as `thumbnail.html`.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
4. Drag-and-drop the file into your browser.
5. Click the `Generate thumbnail` button.

```
<!DOCTYPE html>
<html>
<head>
  <title>Thumbnail Sample</title>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    var subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
    // subscription keys. For example, if you got your subscription keys from
    // westus, replace "westcentralus" in the URI below with "westus".
    //
    // Free trial subscription keys are generated in the westcentralus region.
    // If you use a free trial subscription key, you shouldn't need to change
    // this region.
    var uriBase =
      "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/generateThumbnail";

    // Request parameters.
    var params = "?width=100&height=150&smartCropping=true";

    // Display the source image.
    var sourceImageUrl = document.getElementById("inputImage").value;
    document.querySelector("#sourceImage").src = sourceImageUrl;

    // Prepare the REST API call:
```

```

// Create the HTTP Request object.
var xhr = new XMLHttpRequest();

// Identify the request as a POST, with the URL and parameters.
xhr.open("POST", uriBase + params);

// Add the request headers.
xhr.setRequestHeader("Content-Type","application/json");
xhr.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

// Set the response type to "blob" for the thumbnail image data.
xhr.responseType = "blob";

// Process the result of the REST API call.
xhr.onreadystatechange = function(e) {
    if(xhr.readyState === XMLHttpRequest.DONE) {

        // Thumbnail successfully created.
        if (xhr.status === 200) {
            // Show response headers.
            var s = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);
            document.getElementById("responseTextArea").value =
                JSON.stringify(xhr.getAllResponseHeaders(), null, 2);

            // Show thumbnail image.
            var urlCreator = window.URL || window.webkitURL;
            var imageUrl = urlCreator.createObjectURL(this.response);
            document.querySelector("#thumbnailImage").src = imageUrl;
        } else {
            // Display the error message. The error message is the response
            // body as a JSON string. The code in this code block extracts
            // the JSON string from the blob response.
            var reader = new FileReader();

            // This event fires after the blob has been read.
            reader.addEventListener('loadend', (e) => {
                document.getElementById("responseTextArea").value =
                    JSON.stringify(JSON.parse(e.srcElement.result), null, 2);
            });

            // Start reading the blob as text.
            reader.readAsText(xhr.response);
        }
    }
};

// Make the REST API call.
xhr.send({'url': ' + "'" + sourceImageUrl + "'});
</script>

<h1>Generate thumbnail image:</h1>
Enter the URL to an image to use in creating a thumbnail image,
then click the <strong>Generate thumbnail</strong> button.
<br><br>
Image for thumbnail:
<input type="text" name="inputImage" id="inputImage"
    value="https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Shorkie_Poo_Puppy.jpg/1280px-Shorkie_Poo_Puppy.jpg" />
<button onclick="processImage()">Generate thumbnail</button>
<br><br>
<div id="wrapper" style="width:1160px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput"
            style="width:580px; height:400px;"></textarea>
    </div>

```

```
<div id="imageDiv" style="width:420px; display:table-cell;">
  Source image:
  <br><br>
  <img id="sourceImage" width="400" />
</div>
<div id="thumbnailDiv" style="width:140px; display:table-cell;">
  Thumbnail:
  <br><br>
  <img id="thumbnailImage" />
</div>
</div>
</body>
</html>
```

Get Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong. The following text is an example of a successful response.

Response:

```
"Content-Type: image/jpeg\r\n"
```

Next steps

Explore a JavaScript application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API JavaScript Tutorial](#)

Quickstart: Extract printed text (OCR) with JavaScript

6/15/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following and save it to a file such as `ocr.html`.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
4. Drag-and-drop the file into your browser.
5. Click the `Read image` button.

This sample uses jQuery 1.9.0. For a sample that uses JavaScript without jQuery, see [Intelligently generate a thumbnail](#).

```
<!DOCTYPE html>
<html>
<head>
  <title>OCR Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    var subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
    // subscription keys. For example, if you got your subscription keys from
    // westus, replace "westcentralus" in the URI below with "westus".
    //
    // Free trial subscription keys are generated in the westcentralus region.
    // If you use a free trial subscription key, you shouldn't need to change
    // this region.
    var uriBase =
      "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/ocr";

    // Request parameters.
    var params = {
      "language": "unk",
      "detectOrientation": "true",
```

```

});

// Display the image.
var sourceImageUrl = document.getElementById("inputImage").value;
document.querySelector("#sourceImage").src = sourceImageUrl;

// Perform the REST API call.
$.ajax({
    url: uriBase + "?" + $.param(params),

    // Request headers.
    beforeSend: function(jqXHR){
        jqXHR.setRequestHeader("Content-Type","application/json");
        jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
    },

    type: "POST",

    // Request body.
    data: '{"url": ' + "'" + sourceImageUrl + "'",
})

.done(function(data) {
    // Show formatted JSON on webpage.
    $("#responseTextArea").val(JSON.stringify(data, null, 2));
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Display error message.
    var errorString = (errorThrown === "") ?
        "Error. " : errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" :
        (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message :
            jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
});
</script>

<h1>Optical Character Recognition (OCR)</h1>
Enter the URL to an image of printed text, then
click the <strong>Read image</strong> button.
<br><br>
Image to read:
<input type="text" name="inputImage" id="inputImage"
    value="https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png" />
<button onclick="processImage()">Read image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput"
            style="width:580px; height:400px;"></textarea>
    </div>
    <div id="imageDiv" style="width:420px; display:table-cell;">
        Source image:
        <br><br>
        <img id="sourceImage" width="400" />
    </div>
</div>
</body>
</html>

```

OCR response

A successful response is returned in JSON. The OCR results returned include text, bounding box for regions, lines, and words.

The program produces output similar to the following JSON:

```
{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        },
        {
          "boundingBox": "27,66,283,52",
          "words": [
            {
              "boundingBox": "27,66,283,52",
              "text": "EXISTS"
            }
          ]
        },
        {
          "boundingBox": "27,128,292,49",
          "words": [
            {
              "boundingBox": "27,128,292,49",
              "text": "EXCEPT"
            }
          ]
        },
        {
          "boundingBox": "24,188,292,54",
          "words": [
            {
              "boundingBox": "24,188,292,54",
              "text": "ATOMS"
            }
          ]
        },
        {
          "boundingBox": "22,253,297,32",
          "words": [
            {
              "boundingBox": "22,253,105,32",
              "text": "AND"
            },
            {
              "boundingBox": "144,253,175,32",
              "text": "EMPTY"
            }
          ]
        },
        {
          "boundingBox": "21,298,304,60",
          "words": [
```

```

        {
          "boundingBox": "21,298,304,60",
          "text": "SPACE."
        }
      ],
    },
    {
      "boundingBox": "26,387,294,37",
      "words": [
        {
          "boundingBox": "26,387,210,37",
          "text": "Everything"
        },
        {
          "boundingBox": "249,389,71,27",
          "text": "else"
        }
      ]
    },
  ],
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion."
    }
  ]
}
]
}
]
}
}

```

Next steps

Explore a JavaScript application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API JavaScript Tutorial](#)

Quickstart: Extract handwritten text with JavaScript

6/15/2018 • 4 minutes to read • [Edit Online](#)

In this quickstart, you extract handwritten text from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Recognize Text request

With the [Recognize Text](#) and the [Get Recognize Text Operation Result methods](#), you can detect handwritten text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following and save it to a file such as `handwriting.html`.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you obtained your subscription keys, if necessary.
4. Drag-and-drop the file into your browser.
5. Click the `Read image` button.

This sample uses jQuery 1.9.0. For a sample that uses JavaScript without jQuery, see [Intelligently generate a thumbnail](#).

```
<!DOCTYPE html>
<html>
<head>
  <title>Handwriting Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace <Subscription Key> with your valid subscription key.
    var subscriptionKey = "<Subscription Key>";

    // You must use the same region in your REST call as you used to get your
    // subscription keys. For example, if you got your subscription keys from
    // westus, replace "westcentralus" in the URI below with "westus".
    //
    // Free trial subscription keys are generated in the westcentralus region.
    // If you use a free trial subscription key, you shouldn't need to change
    // this region.
    var uriBase =
      "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/recognizeText";

    // Request parameter.
    // Note: The request parameter changed for APIv2.
    // For APIv1, it is "handwriting": "true".
    var params = {
      "mode": "Handwritten",
```

```

};

// Display the image.
var sourceImageUrl = document.getElementById("inputImage").value;
document.querySelector("#sourceImage").src = sourceImageUrl;

// This operation requires two REST API calls. One to submit the image
// for processing, the other to retrieve the text found in the image.
//
// Make the first REST API call to submit the image for processing.
$.ajax({
    url: uriBase + "?" + $.param(params),

    // Request headers.
    beforeSend: function(jqXHR){
        jqXHR.setRequestHeader("Content-Type","application/json");
        jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
    },

    type: "POST",

    // Request body.
    data: '{"url": ' + "'" + sourceImageUrl + "'",
})

.done(function(data, textStatus, jqXHR) {
    // Show progress.
    $("#responseTextArea").val("Handwritten text submitted. " +
        "Waiting 10 seconds to retrieve the recognized text.");

    // Note: The response may not be immediately available. Handwriting
    // recognition is an asynchronous operation that can take a variable
    // amount of time depending on the length of the text you want to
    // recognize. You may need to wait or retry the GET operation.
    //
    // Wait ten seconds before making the second REST API call.
    setTimeout(function () {
        // "Operation-Location" in the response contains the URI
        // to retrieve the recognized text.
        var operationLocation = jqXHR.getResponseHeader("Operation-Location");

        // Make the second REST API call and get the response.
        $.ajax({
            url: operationLocation,

            // Request headers.
            beforeSend: function(jqXHR){
                jqXHR.setRequestHeader("Content-Type","application/json");
                jqXHR.setRequestHeader(
                    "Ocp-Apim-Subscription-Key", subscriptionKey);
            },

            type: "GET",
        })

        .done(function(data) {
            // Show formatted JSON on webpage.
            $("#responseTextArea").val(JSON.stringify(data, null, 2));
        })

        .fail(function(jqXHR, textStatus, errorThrown) {
            // Display error message.
            var errorString = (errorThrown === "") ? "Error. " :
                errorThrown + " (" + jqXHR.status + "): ";
            errorString += (jqXHR.responseText === "") ? "" :
                (jQuery.parseJSON(jqXHR.responseText).message) ?
                    jQuery.parseJSON(jqXHR.responseText).message :
                    jQuery.parseJSON(jqXHR.responseText).error.message;
            alert(errorString);
        });
    }, 10000);
});

```

```

    });
    }, 10000);
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Put the JSON description into the text area.
    $("#responseTextArea").val(JSON.stringify(jqXHR, null, 2));

    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " :
        errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" :
        (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message :
            jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
});
</script>
<h1>Read handwritten image:</h1>
Enter the URL to an image of handwritten text, then click
the <strong>Read image</strong> button.
<br><br>
Image to read:
<input type="text" name="inputImage" id="inputImage"

value="https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Cursive_Writing_on_Notebook_paper.jpg/800px-Cursive_Writing_on_Notebook_paper.jpg" />
<button onclick="processImage()">Read image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput"
            style="width:580px; height:400px;"></textarea>
    </div>
    <div id="imageDiv" style="width:420px; display:table-cell;">
        Source image:
        <br><br>
        <img id="sourceImage" width="400" />
    </div>
</div>
</body>
</html>

```

Recognize Text response

A successful response is returned in JSON. The handwriting results returned include text, bounding box for regions, lines, and words.

The program produces output similar to the following JSON:

```

{
  "status": "Succeeded",
  "recognitionResult": {
    "lines": [
      {
        "boundingBox": [
          2,
          52,
          65,
          46,
          69,
          89,
          7.

```

```
,
95
],
"text": "dog",
"words": [
  {
    "boundingBox": [
      0,
      59,
      63,
      43,
      77,
      86,
      3,
      102
    ],
    "text": "dog"
  }
]
},
{
  "boundingBox": [
    6,
    2,
    771,
    13,
    770,
    75,
    5,
    64
  ],
  "text": "The quick brown fox jumps over the lazy",
  "words": [
    {
      "boundingBox": [
        0,
        4,
        92,
        5,
        77,
        71,
        0,
        71
      ],
      "text": "The"
    },
    {
      "boundingBox": [
        74,
        4,
        189,
        5,
        174,
        72,
        60,
        71
      ],
      "text": "quick"
    },
    {
      "boundingBox": [
        176,
        5,
        321,
        6,
        306,
        73,
        161,
        72
      ],
      "text": "fox"
    },
    {
      "boundingBox": [
        176,
        5,
        321,
        6,
        306,
        73,
        161,
        72
      ],
      "text": "jumps"
    },
    {
      "boundingBox": [
        176,
        5,
        321,
        6,
        306,
        73,
        161,
        72
      ],
      "text": "over"
    },
    {
      "boundingBox": [
        176,
        5,
        321,
        6,
        306,
        73,
        161,
        72
      ],
      "text": "the"
    },
    {
      "boundingBox": [
        176,
        5,
        321,
        6,
        306,
        73,
        161,
        72
      ],
      "text": "lazy"
    }
  ]
}
```

```
],
  "text": "brown"
},
{
  "boundingBox": [
    308,
    6,
    387,
    6,
    372,
    73,
    293,
    73
  ],
  "text": "fox"
},
{
  "boundingBox": [
    382,
    6,
    506,
    7,
    491,
    74,
    368,
    73
  ],
  "text": "jumps"
},
{
  "boundingBox": [
    492,
    7,
    607,
    8,
    592,
    75,
    478,
    74
  ],
  "text": "over"
},
{
  "boundingBox": [
    589,
    8,
    673,
    8,
    658,
    75,
    575,
    75
  ],
  "text": "the"
},
{
  "boundingBox": [
    660,
    8,
    783,
    9,
    768,
    76,
    645,
    75
  ],
  "text": "lazy"
}
]
```

```

},
{
  "boundingBox": [
    2,
    84,
    783,
    96,
    782,
    154,
    1,
    148
  ],
  "text": "Pack my box with five dozen liquor jugs",
  "words": [
    {
      "boundingBox": [
        0,
        86,
        94,
        87,
        72,
        151,
        0,
        149
      ],
      "text": "Pack"
    },
    {
      "boundingBox": [
        76,
        87,
        164,
        88,
        142,
        152,
        54,
        150
      ],
      "text": "my"
    },
    {
      "boundingBox": [
        155,
        88,
        243,
        89,
        222,
        152,
        134,
        151
      ],
      "text": "box"
    },
    {
      "boundingBox": [
        226,
        89,
        344,
        90,
        323,
        154,
        204,
        152
      ],
      "text": "with"
    },
    {
      "boundingBox": [
        336,

```

```

        90,
        432,
        91,
        411,
        154,
        314,
        154
    ],
    "text": "five"
},
{
    "boundingBox": [
        419,
        91,
        538,
        92,
        516,
        154,
        398,
        154
    ],
    "text": "dozen"
},
{
    "boundingBox": [
        547,
        92,
        701,
        94,
        679,
        154,
        525,
        154
    ],
    "text": "liquor"
},
{
    "boundingBox": [
        696,
        94,
        800,
        95,
        780,
        154,
        675,
        154
    ],
    "text": "jugs"
}
]
}
]
}
}

```

Next steps

Explore a JavaScript application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API JavaScript Tutorial](#)

Quickstart: Analyze an image with Node.js

6/18/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you analyze an image to extract visual features using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze Image request

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you got your subscription keys, if necessary.
4. Optionally, change the `imageUri` value to the image you want to analyze.
5. Optionally, change the response language (`'language': 'en'`).
6. Save the file with an `.js` extension.
7. Open the Node.js command prompt and run the file, for example: `node myfile.js`.

This sample uses the npm [request](#) package.


```

'use strict';

const request = require('request');

// Replace <Subscription Key> with your valid subscription key.
const subscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to get your
// subscription keys. For example, if you got your subscription keys from
// westus, replace "westcentralus" in the URL below with "westus".
const uriBase =
  'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze';

const imageUrl =
  'http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg';

// Request parameters.
const params = {
  'visualFeatures': 'Categories,Description,Color',
  'details': '',
  'language': 'en'
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + '"' + imageUrl + '"}',
  headers: {
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key' : subscriptionKey
  }
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
  let jsonResponse = JSON.stringify(JSON.parse(body), null, ' ');
  console.log('JSON Response\n');
  console.log(jsonResponse);
});

```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "color": {
    "dominantColorForeground": "Grey",
    "dominantColorBackground": "Green",
    "dominantColors": [
      "Grey",
      "Green"
    ],
    "accentColor": "4D5E2F",
    "isBwImg": false
  },
  "requestId": "81b4e400-e3c1-41f1-9020-e6871ad9f0ed",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}

```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Generate a thumbnail with Node.js

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you got your subscription keys, if necessary.
4. Optionally, change the `imageUrl` value to the image you want to analyze.
5. Save the file with an `.js` extension.
6. Open the Node.js command prompt and run the file, for example: `node myfile.js`.

This sample uses the npm [request](#) package.

```

'use strict';

const request = require('request');

// Replace <Subscription Key> with your valid subscription key.
const subscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to get your
// subscription keys. For example, if you got your subscription keys from
// westus, replace "westcentralus" in the URL below with "westus".
const uriBase =
  'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/generateThumbnail';

const imageUrl =
  'https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg';

// Request parameters.
const params = {
  'width': '100',
  'height': '100',
  'smartCropping': 'true'
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + '"' + imageUrl + '"}',
  headers: {
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key' : subscriptionKey
  }
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
});

```

Get Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong.

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Extract printed text (OCR) with Node.js

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uriBase` value to the location where you got your subscription keys, if necessary.
4. Optionally, change the `imageUri` value to the image you want to analyze.
5. Save the file with an `.js` extension.
6. Open the Node.js command prompt and run the file, for example: `node myfile.js`.

This sample uses the npm [request](#) package.

```

'use strict';

const request = require('request');

// Replace <Subscription Key> with your valid subscription key.
const subscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to get your
// subscription keys. For example, if you got your subscription keys from
// westus, replace "westcentralus" in the URL below with "westus".
const uriBase =
  'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/ocr';

const imageUrl = 'https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/' +
  'Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png';

// Request parameters.
const params = {
  'language': 'unk',
  'detectOrientation': 'true',
};

const options = {
  uri: uriBase,
  qs: params,
  body: '{"url": ' + '"' + imageUrl + '"}',
  headers: {
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key' : subscriptionKey
  }
};

request.post(options, (error, response, body) => {
  if (error) {
    console.log('Error: ', error);
    return;
  }
  let jsonResponse = JSON.stringify(JSON.parse(body), null, ' ');
  console.log('JSON Response\n');
  console.log(jsonResponse);
});

```

OCR response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words, for example:

```

{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        }
      ]
    },
    {
      "boundingBox": "27,66,283,52",

```

```
"words": [
  {
    "boundingBox": "27,66,283,52",
    "text": "EXISTS"
  }
],
{
  "boundingBox": "27,128,292,49",
  "words": [
    {
      "boundingBox": "27,128,292,49",
      "text": "EXCEPT"
    }
  ]
},
{
  "boundingBox": "24,188,292,54",
  "words": [
    {
      "boundingBox": "24,188,292,54",
      "text": "ATOMS"
    }
  ]
},
{
  "boundingBox": "22,253,297,32",
  "words": [
    {
      "boundingBox": "22,253,105,32",
      "text": "AND"
    },
    {
      "boundingBox": "144,253,175,32",
      "text": "EMPTY"
    }
  ]
},
{
  "boundingBox": "21,298,304,60",
  "words": [
    {
      "boundingBox": "21,298,304,60",
      "text": "SPACE."
    }
  ]
},
{
  "boundingBox": "26,387,294,37",
  "words": [
    {
      "boundingBox": "26,387,210,37",
      "text": "Everything"
    },
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ]
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
```



```
}  
  "text": "opinion."  
}  
]  
}  
]  
}  
]  
}  
]
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Analyze an image with PHP

6/18/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you analyze an image to extract visual features using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze Image request

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change `uriBase` to use the location where you obtained your subscription keys, if necessary.
4. Optionally, set `imageUri` to the image you want to analyze.
5. Optionally, change the response language (`'language' => 'en'`).
6. Save the file with a `.php` extension.
7. Open the file in a browser window with PHP support.

This sample uses the PHP5 [HTTP_Request2](#) package.

```

<html>
<head>
    <title>Analyze Image Sample</title>
</head>
<body>
<?php
// Replace <Subscription Key> with a valid subscription key.
$ocpApimSubscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to obtain
// your subscription keys. For example, if you obtained your subscription keys
// from westus, replace "westcentralus" in the URL below with "westus".
$uriBase = 'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/';

$imageUrl = 'http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg';

require_once 'HTTP/Request2.php';

$request = new Http_Request2($uriBase . '/analyze');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => $ocpApimSubscriptionKey
);
$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'visualFeatures' => 'Categories,Description',
    'details' => '',
    'language' => 'en'
);
$url->setQueryVariables($parameters);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body parameters
$body = json_encode(array('url' => $imageUrl));

// Request body
$request->setBody($body);

try
{
    $response = $request->send();
    echo "<pre>" .
        json_encode(json_decode($response->getBody()), JSON_PRETTY_PRINT) . "</pre>";
}
catch (HttpException $ex)
{
    echo "<pre>" . $ex . "</pre>";
}
?>
</body>
</html>

```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875,
      "detail": {
        "landmarks": []
      }
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.916458423253597
      }
    ]
  },
  "requestId": "ebf5a1bc-3ba2-4c56-99b4-bbd20ba28705",
  "metadata": {
    "height": 959,
    "width": 1280,
    "format": "Jpeg"
  }
}

```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Use a domain model with PHP

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you use a domain model to identify landmarks or celebrities in an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Recognize Landmark request

With the [Recognize Domain Specific Content method](#), you can identify a specific set of objects in an image. The two domain-specific models that are currently available are *celebrities* and *landmarks*.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change `uriBase` to use the location where you obtained your subscription keys, if necessary.
4. Optionally, set `imageUrl` to the image you want to analyze.
5. Optionally, set `domain` to `celebrities` to use the Celebrities model.
6. Save the file with a `.php` extension.
7. Open the file in a browser window with PHP support.

The following example identifies a landmark in an image.

This sample uses the PHP5 [HTTP_Request2](#) package.

```

<html>
<head>
    <title>Analyze Domain Model Sample</title>
</head>
<body>
<?php
// Replace <Subscription Key> with a valid subscription key.
$ocpApimSubscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to obtain
// your subscription keys. For example, if you obtained your subscription keys
// from westus, replace "westcentralus" in the URL below with "westus".
$uriBase = 'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/';

// Change 'landmarks' to 'celebrities' to use the Celebrities model.
$domain = 'landmarks';

$imageUrl =
    'https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg';

require_once 'HTTP/Request2.php';

$request = new Http_Request2($uriBase . 'models/' . $domain . '/analyze');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => $ocpApimSubscriptionKey
);
$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'model' => $domain
);
$url->setQueryVariables($parameters);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body parameters
$body = json_encode(array('url' => $imageUrl));

// Request body
$request->setBody($body);

try
{
    $response = $request->send();
    echo "<pre>" .
        json_encode(json_decode($response->getBody()), JSON_PRETTY_PRINT) . "</pre>";
}
catch (HttpException $ex)
{
    echo "<pre>" . $ex . "</pre>";
}
?>
</body>
</html>

```

Recognize Landmark response

A successful response is returned in JSON, for example:

```
{
  "result": {
    "landmarks": [
      {
        "name": "Space Needle",
        "confidence": 0.9998177886009216
      }
    ]
  },
  "requestId": "4d26587b-b2b9-408d-a70c-1f8121d84b0d",
  "metadata": {
    "height": 4132,
    "width": 2096,
    "format": "Jpeg"
  }
}
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Generate a thumbnail with PHP

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change `uriBase` to use the location where you obtained your subscription keys, if necessary.
4. Optionally, set `imageUr1` to the image you want to analyze.
5. Save the file with a `.php` extension.
6. Open the file in a browser window with PHP support.

This sample uses the PHP5 [HTTP_Request2](#) package.


```

<html>
<head>
    <title>Get Thumbnail Sample</title>
</head>
<body>
<?php
// Replace <Subscription Key> with a valid subscription key.
$ocpApimSubscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to obtain
// your subscription keys. For example, if you obtained your subscription keys
// from westus, replace "westcentralus" in the URL below with "westus".
$uriBase = 'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/';

$imageUrl =
    'https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg';

require_once 'HTTP/Request2.php';

$request = new Http_Request2($uriBase . 'generateThumbnail');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => $ocpApimSubscriptionKey
);
$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'width' => '100', // Width of the thumbnail.
    'height' => '100', // Height of the thumbnail.
    'smartCropping' => 'true',
);
$url->setQueryVariables($parameters);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body parameters
$body = json_encode(array('url' => $imageUrl));

// Request body
$request->setBody($body);

try
{
    $response = $request->send();
    echo "<pre>" .
        json_encode(json_decode($response->getBody()), JSON_PRETTY_PRINT) . "</pre>";
}
catch (HttpException $ex)
{
    echo "<pre>" . $ex . "</pre>";
}
?>
</body>
</html>

```

Get Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong.

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Extract printed text (OCR) with PHP

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change `uriBase` to use the location where you obtained your subscription keys, if necessary.
4. Optionally, set `imageUri` to the image you want to analyze.
5. Save the file with a `.php` extension.
6. Open the file in a browser window with PHP support.

This sample uses the PHP5 [HTTP_Request2](#) package.

```

<?php
<html>
<head>
    <title>OCR Sample</title>
</head>
<body>
<?php
// Replace <Subscription Key> with a valid subscription key.
$ocpApimSubscriptionKey = '<Subscription Key>';

// You must use the same location in your REST call as you used to obtain
// your subscription keys. For example, if you obtained your subscription keys
// from westus, replace "westcentralus" in the URL below with "westus".
$uriBase = 'https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/';

$imageUrl = 'https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/' .
    'Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png';

require_once 'HTTP/Request2.php';

$request = new Http_Request2($uriBase . 'ocr');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => $ocpApimSubscriptionKey
);
$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'language' => 'unk',
    'detectOrientation' => 'true'
);
$url->setQueryVariables($parameters);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body parameters
$body = json_encode(array('url' => $imageUrl));

// Request body
$request->setBody($body);

try
{
    $response = $request->send();
    echo "<pre>" .
        json_encode(json_decode($response->getBody()), JSON_PRETTY_PRINT) . "</pre>";
}
catch (HttpException $ex)
{
    echo "<pre>" . $ex . "</pre>";
}
?>
</body>
</html>

```

OCR response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words, for example:

```

{
    "language": "en",

```

```
"orientation": "Up",
"textAngle": 0,
"regions": [
  {
    "boundingBox": "21,16,304,451",
    "lines": [
      {
        "boundingBox": "28,16,288,41",
        "words": [
          {
            "boundingBox": "28,16,288,41",
            "text": "NOTHING"
          }
        ]
      },
      {
        "boundingBox": "27,66,283,52",
        "words": [
          {
            "boundingBox": "27,66,283,52",
            "text": "EXISTS"
          }
        ]
      },
      {
        "boundingBox": "27,128,292,49",
        "words": [
          {
            "boundingBox": "27,128,292,49",
            "text": "EXCEPT"
          }
        ]
      },
      {
        "boundingBox": "24,188,292,54",
        "words": [
          {
            "boundingBox": "24,188,292,54",
            "text": "ATOMS"
          }
        ]
      },
      {
        "boundingBox": "22,253,297,32",
        "words": [
          {
            "boundingBox": "22,253,105,32",
            "text": "AND"
          },
          {
            "boundingBox": "144,253,175,32",
            "text": "EMPTY"
          }
        ]
      },
      {
        "boundingBox": "21,298,304,60",
        "words": [
          {
            "boundingBox": "21,298,304,60",
            "text": "SPACE."
          }
        ]
      },
      {
        "boundingBox": "26,387,294,37",
        "words": [
          {
            "boundingBox": "26,387,210,37",
```

```
        "text": "Everything"
      },
      {
        "boundingBox": "249,389,71,27",
        "text": "else"
      }
    ]
  },
  {
    "boundingBox": "127,431,198,36",
    "words": [
      {
        "boundingBox": "127,431,31,29",
        "text": "is"
      },
      {
        "boundingBox": "172,431,153,36",
        "text": "opinion."
      }
    ]
  }
]
}
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Analyze a remote image with Python

6/18/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you analyze a remote image using Computer Vision. To analyze a local image, see [Analyze a local image with Python](#).

You can run this quickstart in a step-by-step fashion using a Jupyter notebook on [MyBinder](#). To launch Binder, select the following button:



Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze a remote image

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Copy the following code to a new Python script file.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `vision_base_url` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the `image_url` value to another image.
5. Run the script.

The following code uses the Python `requests` library to call the Computer Vision Analyze Image API. It returns the results as a JSON object. The API key is passed in via the `headers` dictionary. The types of features to recognize is passed in via the `params` dictionary.

Analyze Image request

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

# You must use the same region in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from
# westus, replace "westcentralus" in the URI below with "westus".
#
# Free trial subscription keys are generated in the westcentralus region.
# If you use a free trial subscription key, you shouldn't need to change
# this region.
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

analyze_url = vision_base_url + "analyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/" + \
    "Broadway_and_Times_Square_by_night.jpg/450px-Broadway_and_Times_Square_by_night.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key }
params = {'visualFeatures': 'Categories,Description,Color'}
data = {'url': image_url}
response = requests.post(analyze_url, headers=headers, params=params, json=data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The most
# relevant caption for the image is obtained from the 'description' property.
analysis = response.json()
print(analysis)
image_caption = analysis["description"]["captions"][0]["text"].capitalize()

# Display the image and overlay it with the caption.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)

```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "outdoor_",
      "score": 0.00390625,
      "detail": {
        "landmarks": []
      }
    },
    {
      "name": "outdoor_street",
      "score": 0.33984375,
      "detail": {
        "landmarks": []
      }
    }
  ],

```



```

"description": {
  "tags": [
    "building",
    "outdoor",
    "street",
    "city",
    "people",
    "busy",
    "table",
    "walking",
    "traffic",
    "filled",
    "large",
    "many",
    "group",
    "night",
    "light",
    "crowded",
    "bunch",
    "standing",
    "man",
    "sign",
    "crowd",
    "umbrella",
    "riding",
    "tall",
    "woman",
    "bus"
  ],
  "captions": [
    {
      "text": "a group of people on a city street at night",
      "confidence": 0.9122243847383961
    }
  ]
},
"color": {
  "dominantColorForeground": "Brown",
  "dominantColorBackground": "Brown",
  "dominantColors": [
    "Brown"
  ],
  "accentColor": "B54316",
  "isBwImg": false
},
"requestId": "c11894eb-de3e-451b-9257-7c8b168073d1",
"metadata": {
  "height": 600,
  "width": 450,
  "format": "Jpeg"
}
}

```

Next steps

Explore a Python application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Python Tutorial](#)

Quickstart: Analyze a local image with Python

6/18/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you analyze a local image using Computer Vision. To analyze a remote image, see [Analyze a remote image with Python](#).

You can run this quickstart in a step-by step fashion using a Jupyter notebook on [MyBinder](#). To launch Binder, select the following button:

launch binder

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze a local image

This sample is similar to [Analyze a remote image with Python](#) except the image to analyze is read locally from disk. Two changes are required:

- Add a `{"Content-Type": "application/octet-stream"}` header to the request.
- Add the image data (byte array) to the body of the request.

To run the sample, do the following steps:

1. Copy the following code to a new Python script file.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `vision_base_url` value to the location where you obtained your subscription keys, if necessary.
4. Change the `image_path` value to the path of a local image.
5. Run the script.

The following code uses the Python `requests` library to call the Computer Vision Analyze Image API. It returns the results as a JSON object. The API key is passed in via the `headers` dictionary. The types of features to recognize is passed in via the `params` dictionary. The binary image data is passed in via the `data` parameter to `requests.post`.

Analyze Image request

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

# You must use the same region in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from
# westus, replace "westcentralus" in the URI below with "westus".
#
# Free trial subscription keys are generated in the westcentralus region.
# If you use a free trial subscription key, you shouldn't need to change
# this region.
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

analyze_url = vision_base_url + "analyze"

# Set image_path to the local path of an image that you want to analyze.
image_path = "C:/Documents/ImageToAnalyze.jpg"

# Read the image into a byte array
image_data = open(image_path, "rb").read()
headers = {'Ocp-Apim-Subscription-Key': subscription_key,
           'Content-Type': 'application/octet-stream'}
params = {'visualFeatures': 'Categories,Description,Color'}
response = requests.post(
    analyze_url, headers=headers, params=params, data=image_data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The most
# relevant caption for the image is obtained from the 'description' property.
analysis = response.json()
print(analysis)
image_caption = analysis["description"]["captions"][0]["text"].capitalize()

# Display the image and overlay it with the caption.
image = Image.open(BytesIO(image_data))
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)

```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "outdoor_",
      "score": 0.00390625,
      "detail": {
        "landmarks": []
      }
    },
    {
      "name": "outdoor_street",
      "score": 0.33984375,
      "detail": {
        "landmarks": []
      }
    }
  ]
}

```

```

    }
  ],
  "description": {
    "tags": [
      "building",
      "outdoor",
      "street",
      "city",
      "people",
      "busy",
      "table",
      "walking",
      "traffic",
      "filled",
      "large",
      "many",
      "group",
      "night",
      "light",
      "crowded",
      "bunch",
      "standing",
      "man",
      "sign",
      "crowd",
      "umbrella",
      "riding",
      "tall",
      "woman",
      "bus"
    ],
    "captions": [
      {
        "text": "a group of people on a city street at night",
        "confidence": 0.9122243847383961
      }
    ]
  },
  "color": {
    "dominantColorForeground": "Brown",
    "dominantColorBackground": "Brown",
    "dominantColors": [
      "Brown"
    ],
    "accentColor": "B54316",
    "isBwImg": false
  },
  "requestId": "c11894eb-de3e-451b-9257-7c8b168073d1",
  "metadata": {
    "height": 600,
    "width": 450,
    "format": "Jpeg"
  }
}

```

Next steps

Explore a Python application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Python Tutorial](#)

Quickstart: Use a domain model with Python

6/18/2018 • 3 minutes to read • [Edit Online](#)

In this quickstart, you use domain models to identify celebrities and landmarks in an image using Computer Vision.

You can run this quickstart in a step-by-step fashion using a Jupyter notebook on [MyBinder](#). To launch Binder, select the following button:

launch binder

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Identify celebrities and landmarks

With the [Recognize Domain Specific Content method](#), you can identify a specific set of objects in an image. The two domain-specific models that are currently available are *celebrities* and *landmarks*.

To run the sample, do the following steps:

1. Copy the following code to a new Python script file.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `vision_base_url` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the `image_url` value to another image.
5. Run the script.

The following code uses the Python `requests` library to call the Computer Vision Analyze Image API. It returns the results as a JSON object. The API key is passed in via the `headers` dictionary. The model to use is passed in via the `params` dictionary.

Landmark identification

Recognize Landmark request

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

# You must use the same region in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from
# westus, replace "westcentralus" in the URI below with "westus".
#
# Free trial subscription keys are generated in the westcentralus region.
# If you use a free trial subscription key, you shouldn't need to change
# this region.
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

landmark_analyze_url = vision_base_url + "models/landmarks/analyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/f/f6/" + \
    "Bunker_Hill_Monument_2005.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'model': 'landmarks'}
data = {'url': image_url}
response = requests.post(
    landmark_analyze_url, headers=headers, params=params, json=data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The
# most relevant landmark for the image is obtained from the 'result' property.
analysis = response.json()
assert analysis["result"]["landmarks"] is not []
print(analysis)
landmark_name = analysis["result"]["landmarks"][0]["name"].capitalize()

# Display the image and overlay it with the landmark name.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(landmark_name, size="x-large", y=-0.1)

```

Recognize Landmark response

A successful response is returned in JSON, for example:

```
{
  "result": {
    "landmarks": [
      {
        "name": "Bunker Hill Monument",
        "confidence": 0.9768505096435547
      }
    ]
  },
  "requestId": "659a10cd-44bb-44db-9147-a295b853b2b8",
  "metadata": {
    "height": 1600,
    "width": 1200,
    "format": "Jpeg"
  }
}
```

Celebrity identification

Recognize Celebrity request

```
import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

celebrity_analyze_url = vision_base_url + "models/celebrities/analyze"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/d/d9/" + \
    "Bill_gates_portrait.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'model': 'celebrities'}
data = {'url': image_url}
response = requests.post(
    celebrity_analyze_url, headers=headers, params=params, json=data)
response.raise_for_status()

# The 'analysis' object contains various fields that describe the image. The
# most relevant celebrity for the image is obtained from the 'result' property.
analysis = response.json()
assert analysis["result"]["celebrities"] is not []
print(analysis)
celebrity_name = analysis["result"]["celebrities"][0]["name"].capitalize()

# Display the image and overlay it with the celebrity name.
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(celebrity_name, size="x-large", y=-0.1)
```

Recognize Celebrity response

A successful response is returned in JSON, for example:

```
{
  "result": {
    "celebrities": [
      {
        "faceRectangle": {
          "top": 123,
          "left": 156,
          "width": 187,
          "height": 187
        },
        "name": "Bill Gates",
        "confidence": 0.9993845224380493
      }
    ]
  },
  "requestId": "f14ec1d0-62d4-4296-9ceb-6b5776dc2020",
  "metadata": {
    "height": 521,
    "width": 550,
    "format": "Jpeg"
  }
}
```

Next steps

Explore a Python application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Python Tutorial](#)

Quickstart: Generate a thumbnail with Python

6/18/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

You can run this quickstart in a step-by-step fashion using a Jupyter notebook on [MyBinder](#). To launch Binder, select the following button:

launch binder

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Intelligently generate a thumbnail

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Copy the following code to a new Python script file.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `vision_base_url` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the `image_url` value to another image.
5. Run the script.

The following code uses the Python `requests` library to call the Computer Vision Analyze Image API. The API key is passed in via the `headers` dictionary. The size of the thumbnail is passed in via the `params` dictionary. The thumbnail is returned as a byte array in the response.

Get Thumbnail request

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

# You must use the same region in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from
# westus, replace "westcentralus" in the URI below with "westus".
#
# Free trial subscription keys are generated in the westcentralus region.
# If you use a free trial subscription key, you shouldn't need to change
# this region.
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

thumbnail_url = vision_base_url + "generateThumbnail"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'width': '50', 'height': '50', 'smartCropping': 'true'}
data = {'url': image_url}
response = requests.post(thumbnail_url, headers=headers, params=params, json=data)
response.raise_for_status()

thumbnail = Image.open(BytesIO(response.content))

# Display the thumbnail.
plt.imshow(thumbnail)
plt.axis("off")

# Verify the thumbnail size.
print("Thumbnail is {0}-by-{1}".format(*thumbnail.size))

```

Next steps

Explore a Python application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Python Tutorial](#)

Quickstart: Extract printed text (OCR) with Python

6/18/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

You can run this quickstart in a step-by-step fashion using a Jupyter notebook on [MyBinder](#). To launch Binder, select the following button:

launch binder

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Extract printed text

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following code to a new Python script file.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `vision_base_url` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the `image_url` value to another image.
5. Run the script.

The following code uses the Python `requests` library to call the Computer Vision Analyze Image API. It returns the results as a JSON object. The API key is passed in via the `headers` dictionary.

OCR request

```

import requests
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

# You must use the same region in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from
# westus, replace "westcentralus" in the URI below with "westus".
#
# Free trial subscription keys are generated in the westcentralus region.
# If you use a free trial subscription key, you shouldn't need to change
# this region.
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

ocr_url = vision_base_url + "ocr"

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/" + \
    "Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'language': 'unk', 'detectOrientation': 'true'}
data = {'url': image_url}
response = requests.post(ocr_url, headers=headers, params=params, json=data)
response.raise_for_status()

analysis = response.json()

# Extract the word bounding boxes and text.
line_infos = [region["lines"] for region in analysis["regions"]]
word_infos = []
for line in line_infos:
    for word_metadata in line:
        for word_info in word_metadata["words"]:
            word_infos.append(word_info)
word_infos

# Display the image and overlay it with the extracted text.
plt.figure(figsize=(5, 5))
image = Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image, alpha=0.5)
for word in word_infos:
    bbox = [int(num) for num in word["boundingBox"].split(",")]
    text = word["text"]
    origin = (bbox[0], bbox[1])
    patch = Rectangle(origin, bbox[2], bbox[3], fill=False, linewidth=2, color='y')
    ax.axes.add_patch(patch)
    plt.text(origin[0], origin[1], text, fontsize=20, weight="bold", va="top")
plt.axis("off")

```

OCR response

A successful response is returned in JSON, for example:

```

{
  "language": "en",
  "orientation": "Up",
  "textAngle": 0,

```

```
"regions": [
  {
    "boundingBox": "21,16,304,451",
    "lines": [
      {
        "boundingBox": "28,16,288,41",
        "words": [
          {
            "boundingBox": "28,16,288,41",
            "text": "NOTHING"
          }
        ]
      },
      {
        "boundingBox": "27,66,283,52",
        "words": [
          {
            "boundingBox": "27,66,283,52",
            "text": "EXISTS"
          }
        ]
      },
      {
        "boundingBox": "27,128,292,49",
        "words": [
          {
            "boundingBox": "27,128,292,49",
            "text": "EXCEPT"
          }
        ]
      },
      {
        "boundingBox": "24,188,292,54",
        "words": [
          {
            "boundingBox": "24,188,292,54",
            "text": "ATOMS"
          }
        ]
      },
      {
        "boundingBox": "22,253,297,32",
        "words": [
          {
            "boundingBox": "22,253,105,32",
            "text": "AND"
          },
          {
            "boundingBox": "144,253,175,32",
            "text": "EMPTY"
          }
        ]
      },
      {
        "boundingBox": "21,298,304,60",
        "words": [
          {
            "boundingBox": "21,298,304,60",
            "text": "SPACE."
          }
        ]
      },
      {
        "boundingBox": "26,387,294,37",
        "words": [
          {
            "boundingBox": "26,387,210,37",
            "text": "Everything"
          }
        ]
      }
    ]
  }
]
```

```
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ],
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion."
    }
  ]
}
]
}
]
```

Next steps

Explore a Python application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Python Tutorial](#)

Quickstart: Extract handwritten text with Python

6/18/2018 • 4 minutes to read • [Edit Online](#)

In this quickstart, you extract handwritten text from an image using Computer Vision.

You can run this quickstart in a step-by step fashion using a Jupyter notebook on [MyBinder](#). To launch Binder, select the following button:

launch binder

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Extract handwritten text

With the [Recognize Text](#) and the [Get Recognize Text Operation Result methods](#), you can detect handwritten text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following code to a new Python script file.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `vision_base_url` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the `image_url` value to another image.
5. Run the script.

The following code uses the Python `requests` library to call the Computer Vision Analyze Image API. It returns the results as a JSON object. The API key is passed in via the `headers` dictionary.

Recognize Text request

```
import requests
import time
# If you are using a Jupyter notebook, uncomment the following line.
#%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from PIL import Image
from io import BytesIO

# Replace <Subscription Key> with your valid subscription key.
subscription_key = "<Subscription Key>"
assert subscription_key

# You must use the same region in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from
# westus, replace "westcentralus" in the URI below with "westus".
#
# Free trial subscription keys are generated in the westcentralus region.
# If you use a free trial subscription key, you shouldn't need to change
# this region.
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/"

text_recognition_url = vision_base_url + "recognizeText"
```

```

# Set image_url to the URL of an image that you want to analyze.
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/" + \
    "Cursive_Writing_on_Notebook_paper.jpg/800px-Cursive_Writing_on_Notebook_paper.jpg"

headers = {'Ocp-Apim-Subscription-Key': subscription_key}
# Note: The request parameter changed for APIv2.
# For APIv1, it is 'handwriting': 'true'.
params = {'mode': 'Handwritten'}
data = {'url': image_url}
response = requests.post(
    text_recognition_url, headers=headers, params=params, json=data)
response.raise_for_status()

# Extracting handwritten text requires two API calls: One call to submit the
# image for processing, the other to retrieve the text found in the image.

# Holds the URI used to retrieve the recognized text.
operation_url = response.headers["Operation-Location"]

# The recognized text isn't immediately available, so poll to wait for completion.
analysis = {}
while "recognitionResult" not in analysis:
    response_final = requests.get(
        response.headers["Operation-Location"], headers=headers)
    analysis = response_final.json()
    time.sleep(1)

# Extract the recognized text, with bounding boxes.
polygons = [(line["boundingBox"], line["text"])
    for line in analysis["recognitionResult"]["lines"]]

# Display the image and overlay it with the extracted text.
plt.figure(figsize=(15, 15))
image = Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image)
for polygon in polygons:
    vertices = [(polygon[0][i], polygon[0][i+1])
        for i in range(0, len(polygon[0]), 2)]
    text = polygon[1]
    patch = Polygon(vertices, closed=True, fill=False, linewidth=2, color='y')
    ax.axes.add_patch(patch)
    plt.text(vertices[0][0], vertices[0][1], text, fontsize=20, va="top")
_ = plt.axis("off")

```

Recognize Text response

A successful response is returned in JSON, for example:

```

{
  "status": "Succeeded",
  "recognitionResult": {
    "lines": [
      {
        "boundingBox": [
          2,
          52,
          65,
          46,
          69,
          89,
          7,
          95
        ],
        "text": "dog",
        "words": [
          {

```



```

    "boundingBox": [
      0,
      59,
      63,
      43,
      77,
      86,
      3,
      102
    ],
    "text": "dog"
  }
]
},
{
  "boundingBox": [
    6,
    2,
    771,
    13,
    770,
    75,
    5,
    64
  ],
  "text": "The quick brown fox jumps over the lazy",
  "words": [
    {
      "boundingBox": [
        0,
        4,
        92,
        5,
        77,
        71,
        0,
        71
      ],
      "text": "The"
    },
    {
      "boundingBox": [
        74,
        4,
        189,
        5,
        174,
        72,
        60,
        71
      ],
      "text": "quick"
    },
    {
      "boundingBox": [
        176,
        5,
        321,
        6,
        306,
        73,
        161,
        72
      ],
      "text": "brown"
    },
    {
      "boundingBox": [
        202

```

```
500,
6,
387,
6,
372,
73,
293,
73
],
"text": "fox"
},
{
  "boundingBox": [
    382,
    6,
    506,
    7,
    491,
    74,
    368,
    73
  ],
  "text": "jumps"
},
{
  "boundingBox": [
    492,
    7,
    607,
    8,
    592,
    75,
    478,
    74
  ],
  "text": "over"
},
{
  "boundingBox": [
    589,
    8,
    673,
    8,
    658,
    75,
    575,
    75
  ],
  "text": "the"
},
{
  "boundingBox": [
    660,
    8,
    783,
    9,
    768,
    76,
    645,
    75
  ],
  "text": "lazy"
}
]
},
{
  "boundingBox": [
    2,
    84,
```

```
183,  
96,  
782,  
154,  
1,  
148  
],  
"text": "Pack my box with five dozen liquor jugs",  
"words": [  
  {  
    "boundingBox": [  
      0,  
      86,  
      94,  
      87,  
      72,  
      151,  
      0,  
      149  
    ],  
    "text": "Pack"  
  },  
  {  
    "boundingBox": [  
      76,  
      87,  
      164,  
      88,  
      142,  
      152,  
      54,  
      150  
    ],  
    "text": "my"  
  },  
  {  
    "boundingBox": [  
      155,  
      88,  
      243,  
      89,  
      222,  
      152,  
      134,  
      151  
    ],  
    "text": "box"  
  },  
  {  
    "boundingBox": [  
      226,  
      89,  
      344,  
      90,  
      323,  
      154,  
      204,  
      152  
    ],  
    "text": "with"  
  },  
  {  
    "boundingBox": [  
      336,  
      90,  
      432,  
      91,  
      411,  
      154,
```

```

        314,
        154
    ],
    "text": "five"
},
{
    "boundingBox": [
        419,
        91,
        538,
        92,
        516,
        154,
        398,
        154
    ],
    "text": "dozen"
},
{
    "boundingBox": [
        547,
        92,
        701,
        94,
        679,
        154,
        525,
        154
    ],
    "text": "liquor"
},
{
    "boundingBox": [
        696,
        94,
        800,
        95,
        780,
        154,
        675,
        154
    ],
    "text": "jugs"
}
]
}
]
}
}

```

Next steps

Explore a Python application that uses Computer Vision to perform optical character recognition (OCR); create smart-cropped thumbnails; plus detect, categorize, tag, and describe visual features, including faces, in an image.

[Computer Vision API Python Tutorial](#)

Quickstart: Analyze an image with Ruby

6/18/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you analyze an image to extract visual features using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Analyze Image request

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uri` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the response language (`'language' => 'en'`).
5. Optionally, change the image (`{"url\":"..."}`) to analyze.
6. Save the file with an `.rb` extension.
7. Open the Ruby Command Prompt and run the file, for example: `ruby myfile.rb`.

```

require 'net/http'

# You must use the same location in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from westus,
# replace "westcentralus" in the URL below with "westus".
uri = URI('https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/analyze')
uri.query = URI.encode_www_form({
  # Request parameters
  'visualFeatures' => 'Categories, Description',
  'details' => 'Landmarks',
  'language' => 'en'
})

request = Net::HTTP::Post.new(uri.request_uri)

# Request headers
# Replace <Subscription Key> with your valid subscription key.
request['Ocp-Apim-Subscription-Key'] = '<Subscription Key>'
request['Content-Type'] = 'application/json'

request.body =
  "{\"url\": \"http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg\"}"

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
  http.request(request)
end

puts response.body

```

Analyze Image response

A successful response is returned in JSON, for example:

```

{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "people_",
      "score": 0.83984375,
      "detail": {
        "celebrities": [
          {
            "name": "Satya Nadella",
            "faceRectangle": {
              "left": 597,
              "top": 162,
              "width": 248,
              "height": 248
            },
            "confidence": 0.999028444
          }
        ]
      }
    }
  ],
  "adult": {
    "isAdultContent": false,
    "isRacyContent": false,
    "adultScore": 0.0934349000453949,
    "racyScore": 0.068613491952419281
  },
  "tags": [

```

```
{
  "name": "person",
  "confidence": 0.98979085683822632
},
{
  "name": "man",
  "confidence": 0.94493889808654785
},
{
  "name": "outdoor",
  "confidence": 0.938492476940155
},
{
  "name": "window",
  "confidence": 0.89513939619064331
}
],
"description": {
  "tags": [
    "person",
    "man",
    "outdoor",
    "window",
    "glasses"
  ],
  "captions": [
    {
      "text": "Satya Nadella sitting on a bench",
      "confidence": 0.48293603002174407
    }
  ]
},
"requestId": "0dbec5ad-a3d3-4f7e-96b4-dfd57efe967d",
"metadata": {
  "width": 1500,
  "height": 1000,
  "format": "Jpeg"
},
"faces": [
  {
    "age": 44,
    "gender": "Male",
    "faceRectangle": {
      "left": 593,
      "top": 160,
      "width": 250,
      "height": 250
    }
  }
],
"color": {
  "dominantColorForeground": "Brown",
  "dominantColorBackground": "Brown",
  "dominantColors": [
    "Brown",
    "Black"
  ],
  "accentColor": "873B59",
  "isBwImg": false
},
"imageType": {
  "clipArtType": 0,
  "lineDrawingType": 0
}
}
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Generate a thumbnail with Ruby

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you generate a thumbnail from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

Get Thumbnail request

With the [Get Thumbnail method](#), you can generate a thumbnail of an image. You specify the height and width, which can differ from the aspect ratio of the input image. Computer Vision uses smart cropping to intelligently identify the region of interest and generate cropping coordinates based on that region.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uri` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the image (`{\"url\": \"...\"}`) to analyze.
5. Save the file with an `.rb` extension.
6. Open the Ruby Command Prompt and run the file, for example: `ruby myfile.rb`.

```
require 'net/http'

# You must use the same location in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from westus,
# replace "westcentralus" in the URL below with "westus".
uri = URI('https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/generateThumbnail')
uri.query = URI.encode_www_form({
  # Request parameters
  'width' => '100',
  'height' => '100',
  'smartCropping' => 'true'
})

request = Net::HTTP::Post.new(uri.request_uri)

# Request headers
# Replace <Subscription Key> with your valid subscription key.
request['Ocp-Apim-Subscription-Key'] = '<Subscription Key>'
request['Content-Type'] = 'application/json'

request.body =
  "{\"url\": \"https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/\" +
  \"Shorkie_Poo_Puppy.jpg/1280px-Shorkie_Poo_Puppy.jpg\"}"

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
  http.request(request)
end

#puts response.body
```

Get Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong.

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Quickstart: Extract printed text (OCR) with Ruby

6/15/2018 • 2 minutes to read • [Edit Online](#)

In this quickstart, you extract printed text, also known as optical character recognition (OCR), from an image using Computer Vision.

Prerequisites

To use Computer Vision, you need a subscription key; see [Obtaining Subscription Keys](#).

OCR request

With the [OCR method](#), you can detect printed text in an image and extract recognized characters into a machine-usable character stream.

To run the sample, do the following steps:

1. Copy the following code into an editor.
2. Replace `<Subscription Key>` with your valid subscription key.
3. Change the `uri` value to the location where you obtained your subscription keys, if necessary.
4. Optionally, change the image (`{\"url\": \"...\"}`) to analyze.
5. Save the file with an `.rb` extension.
6. Open the Ruby Command Prompt and run the file, for example: `ruby myfile.rb`.

```
require 'net/http'

# You must use the same location in your REST call as you used to get your
# subscription keys. For example, if you got your subscription keys from westus,
# replace "westcentralus" in the URL below with "westus".
uri = URI('https://westcentralus.api.cognitive.microsoft.com/vision/v2.0/ocr')
uri.query = URI.encode_www_form({
  # Request parameters
  'language' => 'unk',
  'detectOrientation' => 'true'
})

request = Net::HTTP::Post.new(uri.request_uri)

# Request headers
# Replace <Subscription Key> with your valid subscription key.
request['Ocp-Apim-Subscription-Key'] = '<Subscription Key>'
request['Content-Type'] = 'application/json'

request.body =
  "{\"url\": \"https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/\" +
  \"Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png\"}"

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
  http.request(request)
end

puts response.body
```

OCR response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words, for example:

```
{
  "language": "en",
  "textAngle": -2.0000000000000338,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "462,379,497,258",
      "lines": [
        {
          "boundingBox": "462,379,497,74",
          "words": [
            {
              "boundingBox": "462,379,41,73",
              "text": "A"
            },
            {
              "boundingBox": "523,379,153,73",
              "text": "GOAL"
            },
            {
              "boundingBox": "694,379,265,74",
              "text": "WITHOUT"
            }
          ]
        },
      ],
    },
    {
      "boundingBox": "565,471,289,74",
      "words": [
        {
          "boundingBox": "565,471,41,73",
          "text": "A"
        },
        {
          "boundingBox": "626,471,150,73",
          "text": "PLAN"
        },
        {
          "boundingBox": "801,472,53,73",
          "text": "IS"
        }
      ]
    },
    {
      "boundingBox": "519,563,375,74",
      "words": [
        {
          "boundingBox": "519,563,149,74",
          "text": "JUST"
        },
        {
          "boundingBox": "683,564,41,72",
          "text": "A"
        },
        {
          "boundingBox": "741,564,153,73",
          "text": "WISH"
        }
      ]
    }
  ]
}
```

Next steps

Explore the Computer Vision APIs used to analyze an image, detect celebrities and landmarks, create a thumbnail, and extract printed and handwritten text.

[Explore Computer Vision APIs](#)

Computer Vision API C# Tutorial

5/7/2018 • 4 minutes to read • [Edit Online](#)

Explore a basic Windows application that uses Computer Vision API to perform optical character recognition (OCR), create smart-cropped thumbnails, plus detect, categorize, tag and describe visual features, including faces, in an image. The below example lets you submit an image URL or a locally stored file. You can use this open source example as a template for building your own app for Windows using the Vision API and WPF (Windows Presentation Foundation), a part of .NET Framework.

Prerequisites

Platform requirements

The below example has been developed for the .NET Framework using [Visual Studio 2015, Community Edition](#).

Subscribe to Computer Vision API and get a subscription key

Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services (formerly Project Oxford). For subscription and key management details, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial.

NOTE

The tutorial is designed to use subscription keys in the **westcentralus** region. The subscription keys generated in the Computer Vision free trial use the **westcentralus** region, so they work correctly. If you generated your subscription keys using your Azure account through <https://azure.microsoft.com/>, you must specify the **westcentralus** region. Keys generated outside the **westcentralus** region will not work.

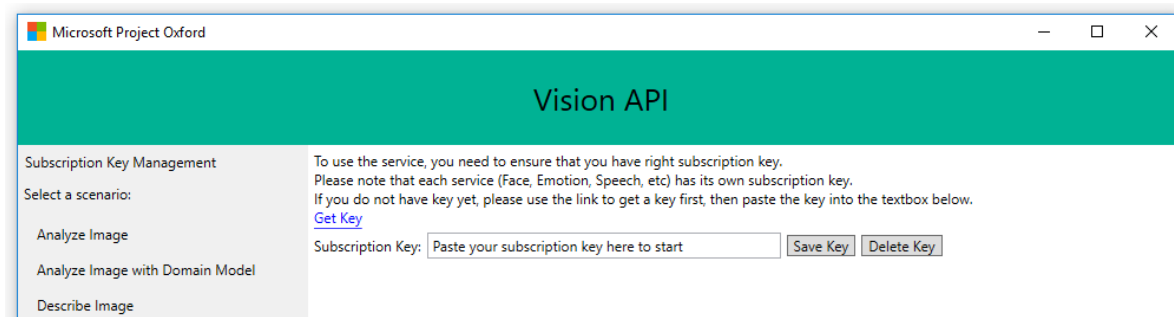
Get the client library and example

You may clone the Computer Vision API client library and example application to your computer via [SDK](#). Don't download it as a ZIP.

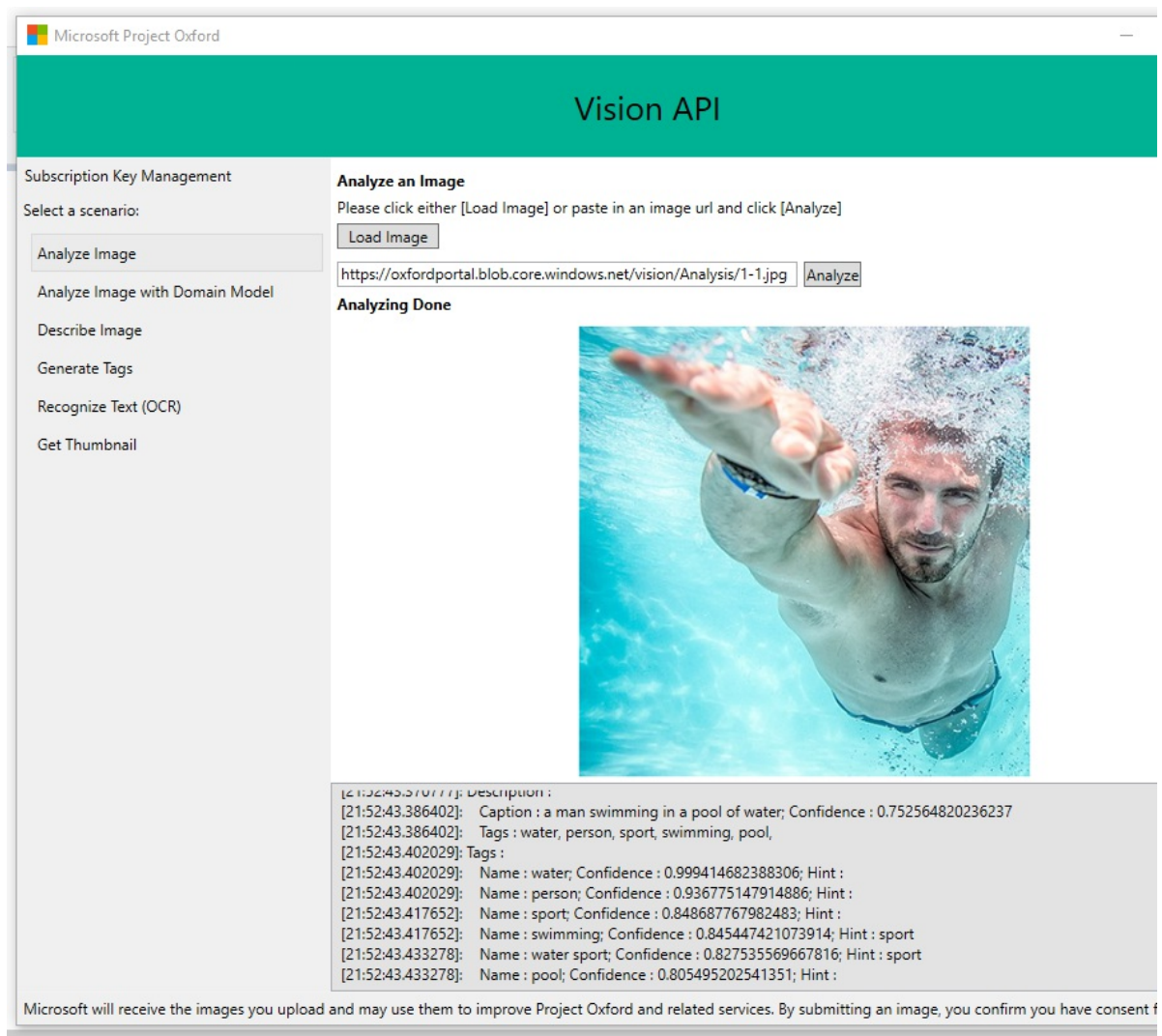
In your GitHub Desktop, open Sample-WPF\VisionAPI-WPF-Samples.sln.

• Press Ctrl+Shift+B, or click Build on the ribbon menu, then select Build Solution.

1. After the build is complete, press **F5** or click **Start** on the ribbon menu to run the example.
2. Locate the Computer Vision API user interface window with the text edit box reading "Paste your subscription key here to start". You can choose to persist your subscription key on your PC or laptop by clicking the "Save Key" button. When you want to delete the subscription key from the system, click "Delete Key" to remove it from your PC or laptop.



3. Under "Select Scenario" click to use one of the six scenarios, then follow the instructions on the screen. Microsoft receives the images you upload and may use them to improve Computer Vision API and related services. By submitting an image, you confirm that you have followed our [Developer Code of Conduct](#).



- There are example images to be used with this example application. You can find these images on the Face API Windows Github repo, in the [Data folder](#). Please note the use of these images is licensed under agreement [LICENSE-IMAGE](#).

Now that you have a running application, let us review how this example app integrates with Cognitive Services technology. This will make it easier to either continue building onto this app or develop your own app using Microsoft Computer Vision API.

This example app makes use of the Computer Vision API Client Library, a thin C# client wrapper for the Microsoft Computer Vision API. When you built the example app as described above, you got the Client Library from a NuGet package. You can review the Client Library source code in the folder titled **"Client Library"** under **Vision, Windows, Client Library**, which is part of the downloaded file repository mentioned above in Prerequisites.

You can also find out how to use the Client Library code in Solution Explorer: Under **VisionAPI-WPF_Samples**, expand **AnalyzePage.xaml** to locate **AnalyzePage.xaml.cs**, which is used for submitting an image to the image analysis endpoint. Double-click the .xaml.cs files to have them open in new windows in Visual Studio.

Reviewing how the Vision Client Library gets used in our example app, let's look at two code snippets from **AnalyzePage.xaml.cs**. The file contains code comments indicating "KEY SAMPLE CODE STARTS HERE" and "KEY SAMPLE CODE ENDS HERE" to help you locate the code snippets reproduced below.

The analyze endpoint is able to work with either an image URL or binary image data (in form of an octet stream) as input. First, you find a using directive, which lets you use the Vision Client Library.

```

// -----
// KEY SAMPLE CODE STARTS HERE
// Use the following namespace for VisionServiceClient
// -----
using Microsoft.ProjectOxford.Vision;
using Microsoft.ProjectOxford.Vision.Contract;
// -----
// KEY SAMPLE CODE ENDS HERE
// -----

```

UploadAndAnalyzeImage(...) This code snippet shows how to use the Client Library to submit your subscription key and a locally stored image to the analyze endpoint of the Computer Vision API service.

```

private async Task<AnalysisResult> UploadAndAnalyzeImage(string imagePath)
{
    // -----
    // KEY SAMPLE CODE STARTS HERE
    // -----
    //
    // Create Project Oxford Computer Vision API Service client
    //
    VisionServiceClient visionServiceClient = new VisionServiceClient(SubscriptionKey);
    Log("VisionServiceClient is created");

    using (Stream imageFileStream = File.OpenRead(imageFilePath))
    {
        //
        // Analyze the image for all visual features
        //
        Log("Calling VisionServiceClient.AnalyzeImageAsync(...)");
        VisualFeature[] visualFeatures = new VisualFeature[] { VisualFeature.Adult, VisualFeature.Categories,
        VisualFeature.Color, VisualFeature.Description, VisualFeature.Faces, VisualFeature.ImageType,
        VisualFeature.Tags };
        AnalysisResult analysisResult = await visionServiceClient.AnalyzeImageAsync(imageFileStream,
        visualFeatures);
        return analysisResult;
    }

    // -----
    // KEY SAMPLE CODE ENDS HERE
    // -----
}

```

AnalyzeUrl(...) This code snippet shows how to use the Client Library to submit your subscription key and a photo URL to the analyze endpoint of the Computer Vision API service.


```

private async Task<AnalysisResult> AnalyzeUrl(string imageUrl)
{
    // -----
    // KEY SAMPLE CODE STARTS HERE
    // -----

    //
    // Create Project Oxford Computer Vision API Service client
    //
    VisionServiceClient visionServiceClient = new VisionServiceClient(SubscriptionKey);
    Log("VisionServiceClient is created");

    //
    // Analyze the url for all visual features
    //
    Log("Calling VisionServiceClient.AnalyzeImageAsync()...");
    VisualFeature[] visualFeatures = new VisualFeature[] { VisualFeature.Adult, VisualFeature.Categories,
    VisualFeature.Color, VisualFeature.Description, VisualFeature.Faces, VisualFeature.ImageType,
    VisualFeature.Tags };
    AnalysisResult analysisResult = await visionServiceClient.AnalyzeImageAsync(imageUrl, visualFeatures);
    return analysisResult;
}

// -----
// KEY SAMPLE CODE ENDS HERE
// -----

```

Other pages and endpoints How to interact with the other endpoints exposed by the Computer Vision API service can be seen by looking at the other pages in the sample; for instance, the OCR endpoint is shown as part of the code contained in `OCRPage.xaml.cs`

🔗 [Get started with Face API](#)

Computer Vision API Java tutorial

3/9/2018 • 22 minutes to read • [Edit Online](#)

This tutorial shows the features of the Microsoft Cognitive Services Computer Vision REST API.

Explore a Java Swing application that uses the Computer Vision REST API to perform optical character recognition (OCR), create smart-cropped thumbnails, plus detect, categorize, tag, and describe visual features, including faces, in an image. This example lets you submit an image URL for analysis or processing. You can use this open source example as a template for building your own app in Java to use the Computer Vision REST API.

This tutorial will cover how to use Computer Vision to:

- Analyze an image
- Identify a natural or artificial landmark in an image
- Identify a celebrity in an image
- Create a quality thumbnail from an image
- Read printed text in an image
- Read handwritten text in an image

The Java Swing form application has already been written, but has no functionality. In this tutorial, you add the code specific to the Computer Vision REST API to complete the application's functionality.

Prerequisites

Platform requirements

This tutorial has been developed using the NetBeans IDE. Specifically, the **Java SE** version of NetBeans, which you can [download here](#).

Subscribe to Computer Vision API and get a subscription key

Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services. For subscription and key management details, see [Subscriptions](#). Both the primary and secondary keys are valid to use in this tutorial.

Download the tutorial project

1. Go to the [Cognitive Services Java Computer Vision Tutorial](#) repository.
2. Click the **Clone or download** button.
3. Click **Download ZIP** to download a .zip file of the tutorial project.

There is no need to extract the contents of the .zip file because NetBeans imports the project from the .zip file.

Import the tutorial project

Import the **cognitive-services-java-computer-vision-tutorial-master.zip** file into NetBeans.

1. In NetBeans, click **File** > **Import Project** > **From ZIP...**. The **Import Project(s) from ZIP** dialog box appears.
2. In the **ZIP File:** field, click the **Browse** button to locate the **cognitive-services-java-computer-vision-tutorial-master.zip** file, then click **Open**.
3. Click **Import** from the **Import Project(s) from ZIP** dialog box.
4. In the **Projects** panel, expand **ComputerVision** > **Source Packages** > **<default package>**. Some versions

of NetBeans use **src** instead of **Source Packages** > **<default package>**. In that case, expand **src**.

5. Double-click **MainFrame.java** to load the file into the NetBeans editor. The **Design** tab of the **MainFrame.java** file appears.
6. Click the **Source** tab to view the Java source code.

Build and run the tutorial project

1. Press **F6** to build and run the tutorial application.

In the tutorial application, click a tab to bring up the pane for that feature. The buttons have empty methods, so they do nothing.

At the bottom of the window are the fields **Subscription Key** and **Subscription Region**. These fields must be filled with a valid subscription key and the correct region for that subscription key. To obtain a subscription key, see [Subscriptions](#). If you obtained your subscription key from the free trial at that link, then the default **westcentralus** is the correct region for your subscription keys.

2. Exit the tutorial application.

Add the tutorial code

The Java Swing application is set up with six tabs. Each tab demonstrates a different function of Computer Vision (analyze, OCR, etc). The six tutorial sections do not have interdependencies, so you can add one section, all six sections, or only a section or two. And you can add the sections in any order.

Let's get started.

Analyze an image

The Analyze feature of Computer Vision analyzes an image for more than 2,000 recognizable objects, living beings, scenery, and actions. Once the analysis is complete, Analyze returns a JSON object that describes the image with descriptive tags, color analysis, captions, and more.

To complete the Analyze feature of the tutorial application, perform the following steps:

Analyze step 1: Add the event handler code for the form button

The **analyzeImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **AnalyzeImage** method to analyze the image. When **AnalyzeImage** returns, the method displays the formatted JSON response in the **Response** text area, extracts the first caption from the **JSONObject**, and displays the caption and the confidence level that the caption is correct.

Copy and paste the following code into the **analyzeImageButtonActionPerformed** method.

NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```

private void analyzeImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL analyzeImageUrl;

    // Clear out the previous image, response, and caption, if any.
    analyzeImage.setIcon(new ImageIcon());
    analyzeCaptionLabel.setText("");
    analyzeResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        analyzeImageUrl = new URL(analyzeImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(analyzeImageUrl);
        scaleAndShowImage(bImage, analyzeImage);
    } catch (IOException e) {
        analyzeResponseTextArea.setText("Error loading Analyze image: " + e.getMessage());
        return;
    }

    // Analyze the image.
    JSONObject jsonObj = AnalyzeImage(analyzeImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    analyzeResponseTextArea.setText(jsonObj.toString(2));

    // Extract the text and confidence from the first caption in the description object.
    if (jsonObj.has("description") && jsonObj.getJSONObject("description").has("captions")) {

        JSONObject jsonCaption =
            jsonObj.getJSONObject("description").getJSONArray("captions").getJSONObject(0);

        if (jsonCaption.has("text") && jsonCaption.has("confidence")) {

            analyzeCaptionLabel.setText("Caption: " + jsonCaption.getString("text") +
                " (confidence: " + jsonCaption.getDouble("confidence") + ").");
        }
    }
}

```

Analyze step 2: Add the wrapper for the REST API call

The **AnalyzeImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** describing the image, or **null** if there was an error.

Copy and paste the **AnalyzeImage** method to just underneath the **analyzeImageButtonActionPerformed** method.

```

/**
 * Encapsulates the Microsoft Cognitive Services REST API call to analyze an image.
 * @param imageUrl: The string URL of the image to analyze.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject AnalyzeImage(String imageUrl) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call for Analyze Image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseAnalyze;
        URIBuilder builder = new URIBuilder(uriString);

        // Request parameters. All of them are optional.
        builder.setParameter("visualFeatures", "Categories,Description,Color,Adult");
        builder.setParameter("language", "en");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}

```

Analyze step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

Recognize a landmark

The Landmark feature of Computer Vision analyzes an image for natural and artificial landmarks, such as mountains or famous buildings. Once the analysis is complete, Landmark returns a JSON object that identifies the landmarks found in the image.

To complete the Landmark feature of the tutorial application, perform the following steps:

Landmark step 1: Add the event handler code for the form button

The **landmarkImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **LandmarkImage** method to analyze the image. When **LandmarkImage** returns, the method displays the formatted JSON response in the **Response** text area, then extracts the first landmark name from the **JSONObject** and displays it on the window along with the confidence level that the landmark was identified correctly.

Copy and paste the following code into the **landmarkImageButtonActionPerformed** method.

NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void landmarkImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL landmarkImageUrl;

    // Clear out the previous image, response, and caption, if any.
    landmarkImage.setIcon(new ImageIcon());
    landmarkCaptionLabel.setText("");
    landmarkResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        landmarkImageUrl = new URL(landmarkImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(landmarkImageUrl);
        scaleAndShowImage(bImage, landmarkImage);
    } catch (IOException e) {
        landmarkResponseTextArea.setText("Error loading Landmark image: " + e.getMessage());
        return;
    }

    // Identify the landmark in the image.
    JSONObject jsonObj = LandmarkImage(landmarkImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    landmarkResponseTextArea.setText(jsonObj.toString(2));

    // Extract the text and confidence from the first caption in the description object.
    if (jsonObj.has("result") && jsonObj.getJSONObject("result").has("landmarks")) {

        JSONObject jsonCaption =
            jsonObj.getJSONObject("result").getJSONArray("landmarks").getJSONObject(0);

        if (jsonCaption.has("name") && jsonCaption.has("confidence")) {

            landmarkCaptionLabel.setText("Caption: " + jsonCaption.getString("name") +
                " (confidence: " + jsonCaption.getDouble("confidence") + ").");
        }
    }
}
```

Landmark step 2: Add the wrapper for the REST API call

The **LandmarkImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** describing the landmarks found in the image, or **null** if there was an error.

Copy and paste the **LandmarkImage** method to just underneath the **landmarkImageButtonActionPerformed** method.

```
/**
 * Encapsulates the Microsoft Cognitive Services REST API call to identify a landmark in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject LandmarkImage(String imageUrl) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to identify a Landmark in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItemAt()) +
            uriBasePostRegion + uriBaseLandmark;
        URIBuilder builder = new URIBuilder(uriString);

        // Request parameters. All of them are optional.
        builder.setParameter("visualFeatures", "Categories,Description,Color");
        builder.setParameter("language", "en");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}
```

Landmark step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Landmark** tab, enter a URL to an image of a landmark, then click the **Analyze Image** button to analyze an image and see the result.

Recognize celebrities

The Celebrities feature of Computer Vision analyzes an image for famous people. Once the analysis is complete, Celebrities returns a JSON object that identifies the Celebrities found in the image.

To complete the Celebrities feature of the tutorial application, perform the following steps:

Celebrities step 1: Add the event handler code for the form button

The **celebritiesImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **CelebritiesImage** method to analyze the image. When **CelebritiesImage** returns, the method displays the formatted JSON response in the **Response** text area, then extracts the first celebrity name from the **JSONObject** and displays the name on the window along with the confidence level that the celebrity was identified correctly.

Copy and paste the following code into the **celebritiesImageButtonActionPerformed** method.

NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.


```

private void celebritiesImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL celebritiesImageUrl;

    // Clear out the previous image, response, and caption, if any.
    celebritiesImage.setIcon(new ImageIcon());
    celebritiesCaptionLabel.setText("");
    celebritiesResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        celebritiesImageUrl = new URL(celebritiesImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(celebritiesImageUrl);
        scaleAndShowImage(bImage, celebritiesImage);
    } catch (IOException e) {
        celebritiesResponseTextArea.setText("Error loading Celebrity image: " + e.getMessage());
        return;
    }

    // Identify the celebrities in the image.
    JSONObject jsonObj = CelebritiesImage(celebritiesImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    celebritiesResponseTextArea.setText(jsonObj.toString(2));

    // Extract the text and confidence from the first caption in the description object.
    if (jsonObj.has("result") && jsonObj.getJSONObject("result").has("celebrities")) {

        JSONObject jsonCaption =
        jsonObj.getJSONObject("result").getJSONArray("celebrities").getJSONObject(0);

        if (jsonCaption.has("name") && jsonCaption.has("confidence")) {

            celebritiesCaptionLabel.setText("Caption: " + jsonCaption.getString("name") +
            " (confidence: " + jsonCaption.getDouble("confidence") + ").");
        }
    }
}

```

Celebrities step 2: Add the wrapper for the REST API call

The **CelebritiesImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** describing the celebrities found in the image, or **null** if there was an error.

Copy and paste the **CelebritiesImage** method to just underneath the **celebritiesImageButtonActionPerformed** method.

```

/**
 * Encapsulates the Microsoft Cognitive Services REST API call to identify celebrities in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject CelebritiesImage(String imageUrl) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to identify celebrities in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseCelebrities;
        URIBuilder builder = new URIBuilder(uriString);

        // Request parameters. All of them are optional.
        builder.setParameter("visualFeatures", "Categories,Description,Color");
        builder.setParameter("language", "en");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}

```

Celebrities step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Celebrities** tab, enter a URL to an image of a celebrity, then click the **Analyze Image** button to analyze an image and see the result.

Intelligently generate a thumbnail

The Thumbnail feature of Computer Vision generates a thumbnail from an image. By using the **Smart Crop** feature, the Thumbnail feature will identify the area of interest in an image and center the thumbnail on this area, to generate more aesthetically pleasing thumbnail images.

To complete the Thumbnail feature of the tutorial application, perform the following steps:

Thumbnail step 1: Add the event handler code for the form button

The **thumbnailImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **getThumbnailImage** method to create the thumbnail. When **getThumbnailImage** returns, the method displays the generated thumbnail.

Copy and paste the following code into the **thumbnailImageButtonActionPerformed** method.

NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void thumbnailImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL thumbnailImageUrl;
    JSONObject jsonError[] = new JSONObject[1];

    // Clear out the previous image, response, and thumbnail, if any.
    thumbnailSourceImage.setIcon(new ImageIcon());
    thumbnailResponseTextArea.setText("");
    thumbnailImage.setIcon(new ImageIcon());

    // Display the image specified in the text box.
    try {
        thumbnailImageUrl = new URL(thumbnailImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(thumbnailImageUrl);
        scaleAndShowImage(bImage, thumbnailSourceImage);
    } catch (IOException e) {
        thumbnailResponseTextArea.setText("Error loading image to thumbnail: " + e.getMessage());
        return;
    }

    // Get the thumbnail for the image.
    BufferedImage thumbnail = getThumbnailImage(thumbnailImageUrl.toString(), jsonError);

    // A non-null value indicates error.
    if (jsonError[0] != null) {
        // Format and display the JSON error.
        thumbnailResponseTextArea.setText(jsonError[0].toString(2));
        return;
    }

    // Display the thumbnail.
    if (thumbnail != null) {
        scaleAndShowImage(thumbnail, thumbnailImage);
    }
}
```

Thumbnail step 2: Add the wrapper for the REST API call

The **getThumbnailImage** method wraps the REST API call to analyze an image. The method returns a **BufferedImage** that contains the thumbnail, or **null** if there was an error. The error message will be returned in the first element of the **jsonError** string array.

Copy and paste the following **getThumbnailImage** method to just underneath the **thumbnailImageButtonActionPerformed** method.

```

/**
 * Encapsulates the Microsoft Cognitive Services REST API call to create a thumbnail for an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A BufferedImage containing the thumbnail, or null if a runtime error occurs. In the case
 * of an error, the error message will be returned in the first element of the jsonError string array.
 */
private BufferedImage getThumbnailImage(String imageUrl, JSONObject[] jsonError) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to identify celebrities in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseThumbnail;
        URIBuilder uriBuilder = new URIBuilder(uriString);

        // Request parameters.
        uriBuilder.setParameter("width", "100");
        uriBuilder.setParameter("height", "150");
        uriBuilder.setParameter("smartCropping", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity requestEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(requestEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // Check for success.
        if (response.getStatusLine().getStatusCode() == 200)
        {
            // Return the thumbnail.
            return ImageIO.read(entity.getContent());
        }
        else
        {
            // Format and display the JSON error message.
            String jsonString = EntityUtils.toString(entity);
            jsonError[0] = new JSONObject(jsonString);
            return null;
        }
    }
    catch (Exception e)
    {
        String errorMessage = e.getMessage();
        System.out.println(errorMessage);
        jsonError[0] = new JSONObject(errorMessage);
        return null;
    }
}

```

Thumbnail step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Thumbnail** tab, enter a URL to an image, then click the **Generate Thumbnail** button to analyze an image and see the result.

Read printed text (OCR)

The Optical Character Recognition (OCR) feature of Computer Vision analyzes an image of printed text. After the analysis is complete, OCR returns a JSON object that contains the text and the location of the text in the image.

To complete the OCR feature of the tutorial application, perform the following steps:

OCR step 1: Add the event handler code for the form button

The **ocrImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **OcrImage** method to analyze the image. When **OcrImage** returns, the method displays the detected text as formatted JSON in the **Response** text area.

Copy and paste the following code into the **ocrImageButtonActionPerformed** method.

NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void ocrImageButtonActionPerformed(java.awt.event.ActionEvent evt) {  
    URL ocrImageUrl;  
  
    // Clear out the previous image, response, and caption, if any.  
    ocrImage.setIcon(new ImageIcon());  
    ocrResponseTextArea.setText("");  
  
    // Display the image specified in the text box.  
    try {  
        ocrImageUrl = new URL(ocrImageUriTextBox.getText());  
        BufferedImage bImage = ImageIO.read(ocrImageUrl);  
        scaleAndShowImage(bImage, ocrImage);  
    } catch (IOException e) {  
        ocrResponseTextArea.setText("Error loading OCR image: " + e.getMessage());  
        return;  
    }  
  
    // Read the text in the image.  
    JSONObject jsonObj = OcrImage(ocrImageUrl.toString());  
  
    // A return of null indicates failure.  
    if (jsonObj == null) {  
        return;  
    }  
  
    // Format and display the JSON response.  
    ocrResponseTextArea.setText(jsonObj.toString(2));  
}
```

OCR step 2: Add the wrapper for the REST API call

The **OcrImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** of the JSON data returned from the call, or **null** if there was an error.

Copy and paste the following **OcrImage** method to just underneath the **ocrImageButtonActionPerformed** method.

```

/**
 * Encapsulates the Microsoft Cognitive Services REST API call to read text in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject OcrImage(String imageUrl) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to read text in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseOcr;
        URIBuilder uriBuilder = new URIBuilder(uriString);

        // Request parameters.
        uriBuilder.setParameter("language", "unk");
        uriBuilder.setParameter("detectOrientation ", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}

```

OCR step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **OCR** tab, enter a URL to an image of printed text, then click the **Read Image** button to analyze an image and see the result.

Read handwritten text (handwriting recognition)

The Handwriting Recognition feature of Computer Vision analyzes an image of handwritten text. After the analysis is complete, Handwriting Recognition returns a JSON object that contains the text and the location of the text in the image.

To complete the Handwriting Recognition feature of the tutorial application, perform the following steps:

Handwriting Recognition step 1: Add the event handler code for the form button

The **handwritingImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **HandwritingImage** method to analyze the image. When **HandwritingImage** returns, the method displays the detected text as formatted JSON in the **Response** text area.

Copy and paste the following code into the **handwritingImageButtonActionPerformed** method.

NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void handwritingImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL handwritingImageUrl;

    // Clear out the previous image, response, and caption, if any.
    handwritingImage.setIcon(new ImageIcon());
    handwritingResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        handwritingImageUrl = new URL(handwritingImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(handwritingImageUrl);
        scaleAndShowImage(bImage, handwritingImage);
    } catch (IOException e) {
        handwritingResponseTextArea.setText("Error loading Handwriting image: " + e.getMessage());
        return;
    }

    // Read the text in the image.
    JSONObject jsonObj = HandwritingImage(handwritingImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    handwritingResponseTextArea.setText(jsonObj.toString(2));
}
```

Handwriting Recognition step 2: Add the wrapper for the REST API call

The **HandwritingImage** method wraps the two REST API calls needed to analyze an image. Because handwriting recognition is a time consuming process, a two step process is used. The first call submits the image for processing; the second call retrieves the detected text when the processing is complete.

After the text is retrieved, the **HandwritingImage** method returns a **JSONObject** describing the text and the locations of the text, or **null** if there was an error.

Copy and paste the following **HandwritingImage** method to just underneath the **handwritingImageButtonActionPerformed** method.

```
/**
 * Encapsulates the Microsoft Cognitive Services REST API call to read handwritten text in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
```

```

private JSONObject HandwritingImage(String imageUrl) {
    try (CloseableHttpClient textClient = HttpClientBuilder.create().build();
        CloseableHttpClient resultClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to read text in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseHandwriting;
        URIBuilder uriBuilder = new URIBuilder(uriString);

        // Request parameters.
        uriBuilder.setParameter("handwriting", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response.
        HttpResponse textResponse = textClient.execute(request);

        // Check for success.
        if (textResponse.getStatusLine().getStatusCode() != 202) {
            // An error occurred. Return the JSON error message.
            HttpEntity entity = textResponse.getEntity();
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        }

        String operationLocation = null;

        // The 'Operation-Location' in the response contains the URI to retrieve the recognized text.
        Header[] responseHeaders = textResponse.getAllHeaders();
        for(Header header : responseHeaders) {
            if(header.getName().equals("Operation-Location"))
            {
                // This string is the URI where you can get the text recognition operation result.
                operationLocation = header.getValue();
                break;
            }
        }

        // NOTE: The response may not be immediately available. Handwriting recognition is an
        // async operation that can take a variable amount of time depending on the length
        // of the text you want to recognize. You may need to wait or retry this operation.
        //
        // This example checks once per second for ten seconds.

        JSONObject responseObj = null;
        int i = 0;
        do {
            // Wait one second.
            Thread.sleep(1000);

            // Check to see if the operation completed.
            HttpGet resultRequest = new HttpGet(operationLocation);
            resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());
            HttpResponse resultResponse = resultClient.execute(resultRequest);
            HttpEntity responseEntity = resultResponse.getEntity();
            if (responseEntity != null)
            {
                // Get the JSON response.
            }
        } while (i < 10);
    }
}

```



```

        String jsonString = EntityUtils.toString(responseEntity);
        responseObj = new JSONObject(jsonString);
    }
}
while (i < 10 && responseObj != null &&
        !responseObj.getString("status").equalsIgnoreCase("Succeeded"));

// If the operation completed, return the JSON object.
if (responseObj != null) {
    return responseObj;
} else {
    // Return null for timeout error.
    System.out.println("Timeout error.");
    return null;
}
}
catch (Exception e)
{
    // Display error message.
    System.out.println(e.getMessage());
    return null;
}
}

```

Handwriting Recognition step 3: Run the application

To run the application, press **F6**. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Read Handwritten Text** tab, enter a URL to an image of handwritten text, then click the **Read Image** button to analyze an image and see the result.

Next steps

- [Computer Vision API C# Tutorial](#)
- [Computer Vision API Python Tutorial](#)

Computer Vision API JavaScript tutorial

4/19/2018 • 18 minutes to read • [Edit Online](#)

This tutorial shows the features of the Microsoft Cognitive Services Computer Vision REST API.

Explore a JavaScript application that uses the Computer Vision REST API to perform optical character recognition (OCR), create smart-cropped thumbnails, plus detect, categorize, tag, and describe visual features, including faces, in an image. This example lets you submit an image URL for analysis or processing. You can use this open source example as a template for building your own app in JavaScript to use the Computer Vision REST API.

The JavaScript form application has already been written, but has no Computer Vision functionality. In this tutorial, you add the code specific to the Computer Vision REST API to complete the application's functionality.

Prerequisites

Platform requirements

This tutorial has been developed using a simple text editor.

Subscribe to Computer Vision API and get a subscription key

Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services. For subscription and key management details, see [Subscriptions](#). Both the primary and secondary keys are valid to use in this tutorial.

Download the tutorial project

Clone the [Cognitive Services JavaScript Computer Vision Tutorial](#), or download the .zip file and extract it to an empty directory.

If you would prefer to use the finished tutorial with all tutorial code added, you can use the files in the **Completed** folder.

Add the tutorial code

The JavaScript application is set up with six .html files, one for each feature. Each file demonstrates a different function of Computer Vision (analyze, OCR, etc). The six tutorial sections do not have interdependencies, so you can add the tutorial code to one file, all six files, or only a couple of files. And you can add the tutorial code to the files in any order.

Let's get started.

Analyze an image

The Analyze feature of Computer Vision analyzes an image for more than 2,000 recognizable objects, living beings, scenery, and actions. Once the analysis is complete, Analyze returns a JSON object that describes the image with descriptive tags, color analysis, captions, and more.

To complete the Analyze feature of the tutorial application, perform the following steps:

Analyze step 1: Add the event handler code for the form button

Open the **analyze.html** file in a text editor and locate the **analyzeButtonClick** function near the bottom of the file.

The **analyzeButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **AnalyzeImage** function to analyze the image.

Copy and paste the following code into the **analyzeButtonClick** function.

```
function analyzeButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    AnalyzeImage(sourceImageUrl, $("#responseTextArea"), $("#captionSpan"));  
}
```

Analyze step 2: Add the wrapper for the REST API call

The **AnalyzeImage** function wraps the REST API call to analyze an image. Upon a successful return, the formatted JSON analysis will be displayed in the specified textarea, and the caption will be displayed in the specified span.

Copy and paste the **AnalyzeImage** function code to just underneath the **analyzeButtonClick** function.

```

/* Analyze the image at the specified URL by using Microsoft Cognitive Services Analyze Image API.
 * @param {string} sourceImageUrl - The URL to the image to analyze.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 * @param {<span> element} captionSpan - The span to display the image caption.
 */
function AnalyzeImage(sourceImageUrl, responseTextArea, captionSpan) {
    // Request parameters.
    var params = {
        "visualFeatures": "Categories,Description,Color",
        "details": "",
        "language": "en",
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseAnalyze +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR){
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'",
    });

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));

        // Extract and display the caption and confidence from the first caption in the description object.
        if (data.description && data.description.captions) {
            var caption = data.description.captions[0];

            if (caption.text && caption.confidence) {
                captionSpan.text("Caption: " + caption.text +
                    " (confidence: " + caption.confidence + ").");
            }
        }
    });

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Prepare the error string.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;

        // Put the error JSON in the response textarea.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Show the error message.
        alert(errorString);
    });
}

```

Analyze step 3: Run the application

Save the **analyze.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

Recognize a landmark

The Landmark feature of Computer Vision analyzes an image for natural and artificial landmarks, such as mountains or famous buildings. Once the analysis is complete, Landmark returns a JSON object that identifies the landmarks found in the image.

To complete the Landmark feature of the tutorial application, perform the following steps:

Landmark step 1: Add the event handler code for the form button

Open the **landmark.html** file in a text editor and locate the **landmarkButtonClick** function near the bottom of the file.

The **landmarkButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **IdentifyLandmarks** function to analyze the image.

Copy and paste the following code into the **landmarkButtonClick** function.

```
function landmarkButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    IdentifyLandmarks(sourceImageUrl, $("#responseTextArea"), $("#captionSpan"));  
}
```

Landmark step 2: Add the wrapper for the REST API call

The **IdentifyLandmarks** function wraps the REST API call to analyze an image. Upon a successful return, the formatted JSON analysis will be displayed in the specified textarea, and the caption will be displayed in the specified span.

Copy and paste the **IdentifyLandmarks** function code to just underneath the **landmarkButtonClick** function.

```

/* Identify landmarks in the image at the specified URL by using Microsoft Cognitive Services
 * Landmarks API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for landmarks.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 * @param {<span> element} captionSpan - The span to display the image caption.
 */
function IdentifyLandmarks(sourceImageUrl, responseTextArea, captionSpan) {
    // Request parameters.
    var params = {
        "model": "landmarks"
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseLandmark +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR){
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'",
    });

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));

        // Extract and display the caption and confidence from the first caption in the description object.
        if (data.result && data.result.landmarks) {
            var landmark = data.result.landmarks[0];

            if (landmark.name && landmark.confidence) {
                captionSpan.text("Landmark: " + landmark.name +
                    " (confidence: " + landmark.confidence + ").");
            }
        }
    });

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Prepare the error string.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;

        // Put the error JSON in the response textarea.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Show the error message.
        alert(errorString);
    });
}

```

Landmark step 3: Run the application

Save the **landmark.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

Recognize celebrities

The Celebrities feature of Computer Vision analyzes an image for famous people. Once the analysis is complete, Celebrities returns a JSON object that identifies the Celebrities found in the image.

To complete the Celebrities feature of the tutorial application, perform the following steps:

Celebrities step 1: Add the event handler code for the form button

Open the **celebrities.html** file in a text editor and locate the **celebritiesButtonClick** function near the bottom of the file.

The **celebritiesButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **IdentifyCelebrities** function to analyze the image.

Copy and paste the following code into the **celebritiesButtonClick** function.

```
function celebritiesButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    IdentifyCelebrities(sourceImageUrl, $("#responseTextArea"), $("#captionSpan"));  
}
```

Celebrities step 2: Add the wrapper for the REST API call

```

/* Identify celebrities in the image at the specified URL by using Microsoft Cognitive Services
 * Celebrities API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for celebrities.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 * @param {<span> element} captionSpan - The span to display the image caption.
 */
function IdentifyCelebrities(sourceImageUrl, responseTextArea, captionSpan) {
    // Request parameters.
    var params = {
        "model": "celebrities"
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseCelebrities +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR){
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'",
    });

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));

        // Extract and display the caption and confidence from the first caption in the description object.
        if (data.result && data.result.celebrities) {
            var celebrity = data.result.celebrities[0];

            if (celebrity.name && celebrity.confidence) {
                captionSpan.text("Celebrity name: " + celebrity.name +
                    " (confidence: " + celebrity.confidence + ").");
            }
        }
    });

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Prepare the error string.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;

        // Put the error JSON in the response textarea.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Show the error message.
        alert(errorString);
    });
}

```

Celebrities step 3: Run the application

Save the **celebrities.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

Intelligently generate a thumbnail

The Thumbnail feature of Computer Vision generates a thumbnail from an image. By using the **Smart Crop** feature, the Thumbnail feature will identify the area of interest in an image and center the thumbnail on this area, to generate more aesthetically pleasing thumbnail images.

To complete the Thumbnail feature of the tutorial application, perform the following steps:

Thumbnail step 1: Add the event handler code for the form button

Open the **thumbnail.html** file in a text editor and locate the **thumbnailButtonClick** function near the bottom of the file.

The **thumbnailButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **getThumbnail** function twice to create two thumbnails, one smart cropped and one without smart crop.

Copy and paste the following code into the **thumbnailButtonClick** function.

```
function thumbnailButtonClick() {

    // Clear the display fields.
    document.getElementById("sourceImage").src = "#";
    document.getElementById("thumbnailImageSmartCrop").src = "#";
    document.getElementById("thumbnailImageNonSmartCrop").src = "#";
    document.getElementById("responseTextArea").value = "";
    document.getElementById("captionSpan").text = "";

    // Display the image.
    var sourceImageUrl = document.getElementById("inputImage").value;
    document.getElementById("sourceImage").src = sourceImageUrl;

    // Get a smart cropped thumbnail.
    getThumbnail (sourceImageUrl, true, document.getElementById("thumbnailImageSmartCrop"),
        document.getElementById("responseTextArea"));

    // Get a non-smart-cropped thumbnail.
    getThumbnail (sourceImageUrl, false, document.getElementById("thumbnailImageNonSmartCrop"),
        document.getElementById("responseTextArea"));
}
```

Thumbnail step 2: Add the wrapper for the REST API call

The **getThumbnail** function wraps the REST API call to analyze an image. Upon a successful return, the thumbnail will be displayed in the specified img element.

Copy and paste the following **getThumbnail** function to just underneath the **thumbnailButtonClick** function.

```
/* Get a thumbnail of the image at the specified URL by using Microsoft Cognitive Services
 * Thumbnail API.
 * @param {string} sourceImageUrl URL to image.
 * @param {boolean} smartCropping Set to true to use the smart cropping feature which crops to the
 * more interesting area of an image; false to crop for the center
 * of the image.
 * @param {<img> element} imageElement The img element in the DOM which will display the thumbnail image.
 * @param {<textarea> element} responseTextArea - The text area to display the Response Headers returned
 * from the REST API call, or to display the error message if there was
 * an error.
 */
function getThumbnail (sourceImageUrl, smartCropping, imageElement, responseTextArea) {
```

```

// Create the HTTP Request object.
var xhr = new XMLHttpRequest();

// Request parameters.
var params = "width=100&height=150&smartCropping=" + smartCropping.toString();

// Build the full URI.
var fullUri = common.uriBasePreRegion +
    document.getElementById("subscriptionRegionSelect").value +
    common.uriBasePostRegion +
    common.uriBaseThumbnail +
    "?" +
    params;

// Identify the request as a POST, with the URI and parameters.
xhr.open("POST", fullUri);

// Add the request headers.
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Ocp-Apim-Subscription-Key",
    encodeURIComponent(document.getElementById("subscriptionKeyInput").value));

// Set the response type to "blob" for the thumbnail image data.
xhr.responseType = "blob";

// Process the result of the REST API call.
xhr.onreadystatechange = function(e) {
    if(xhr.readyState === XMLHttpRequest.DONE) {

        // Thumbnail successfully created.
        if (xhr.status === 200) {
            // Show response headers.
            var s = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);
            responseTextArea.value = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);

            // Show thumbnail image.
            var urlCreator = window.URL || window.webkitURL;
            var imageUrl = urlCreator.createObjectURL(this.response);
            imageElement.src = imageUrl;
        } else {
            // Display the error message. The error message is the response body as a JSON string.
            // The code in this code block extracts the JSON string from the blob response.
            var reader = new FileReader();

            // This event fires after the blob has been read.
            reader.addEventListener('loadend', (e) => {
                responseTextArea.value = JSON.stringify(JSON.parse(e.srcElement.result), null, 2);
            });

            // Start reading the blob as text.
            reader.readAsText(xhr.response);
        }
    }
}

// Execute the REST API call.
xhr.send({'url': ' + "'" + sourceImageUrl + "'});
}

```

Thumbnail step 3: Run the application

Save the **thumbnail.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Generate Thumbnails** button to analyze an image and see the result.

Read printed text (OCR)

The Optical Character Recognition (OCR) feature of Computer Vision analyzes an image of printed text. After the analysis is complete, OCR returns a JSON object that contains the text and the location of the text in the image.

To complete the OCR feature of the tutorial application, perform the following steps:

OCR step 1: Add the event handler code for the form button

Open the **ocr.html** file in a text editor and locate the **ocrButtonClick** function near the bottom of the file.

The **ocrButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **ReadOcrImage** function to analyze the image.

Copy and paste the following code into the **ocrButtonClick** function.

```
function ocrButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "#");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    ReadOcrImage(sourceImageUrl, $("#responseTextArea"));  
}
```

OCR step 2: Add the wrapper for the REST API call

The **ReadOcrImage** function wraps the REST API call to analyze an image. Upon a successful return, the formatted JSON describing the text and the location of the text will be displayed in the specified textarea.

Copy and paste the following **ReadOcrImage** function to just underneath the **ocrButtonClick** function.

```

/* Recognize and read printed text in an image at the specified URL by using Microsoft Cognitive
 * Services OCR API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for printed text.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 */
function ReadOcrImage(sourceImageUrl, responseTextArea) {
    // Request parameters.
    var params = {
        "language": "unk",
        "detectOrientation ": "true",
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseOcr +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR){
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'",
    })

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Put the JSON description into the text area.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Display error message.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
        alert(errorString);
    });
}

```

OCR step 3: Run the application

Save the **ocr.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image of text to read, then click the **Read Image** button to analyze an image and see the result.

Read handwritten text (Handwriting Recognition)

The Handwriting Recognition feature of Computer Vision analyzes an image of handwritten text. After the analysis is complete, Handwriting Recognition returns a JSON object that contains the text and the location of the text in the image.

To complete the Handwriting Recognition feature of the tutorial application, perform the following steps:

Handwriting Recognition step 1: Add the event handler code for the form button

Open the **handwriting.html** file in a text editor and locate the **handwritingButtonClick** function near the bottom of the file.

The **handwritingButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **ReadHandwrittenImage** function to analyze the image.

Copy and paste the following code into the **handwritingButtonClick** function.

```
function handwritingButtonClick() {

    // Clear the display fields.
    $("#sourceImage").attr("src", "");
    $("#responseTextArea").val("");

    // Display the image.
    var sourceImageUrl = $("#inputImage").val();
    $("#sourceImage").attr("src", sourceImageUrl);

    ReadHandwrittenImage(sourceImageUrl, $("#responseTextArea"));
}
```

Handwriting Recognition step 2: Add the wrapper for the REST API call

The **ReadHandwrittenImage** function wraps the two REST API calls needed to analyze an image. Because Handwriting Recognition is a time consuming process, a two step process is used. The first call submits the image for processing; the second call retrieves the detected text when the processing is complete.

After the text is retrieved, the formatted JSON describing the text and the location of the text will be displayed in the specified textarea.

Copy and paste the following **ReadHandwrittenImage** function to just underneath the **handwritingButtonClick** function.

```
/* Recognize and read text from an image of handwriting at the specified URL by using Microsoft
 * Cognitive Services Recognize Handwritten Text API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for handwriting.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 */
function ReadHandwrittenImage(sourceImageUrl, responseTextArea) {
    // Request parameters.
    var params = {
        "handwriting": "true",
    };

    // This operation requires two REST API calls. One to submit the image for processing,
    // the other to retrieve the text found in the image.
    //
    // Perform the first REST API call to submit the image for processing.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseHandwriting +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR){
            jqXHR.setRequestHeader("Content-Type", "application/json");
        }
    });
}
```

```

jqXHR.setRequestHeader("Content-Type", "application/json");
jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
    encodeURIComponent($("#subscriptionKeyInput").val()));
},

type: "POST",

// Request body.
data: '{"url": ' + "'" + sourceImageUrl + "'",
})

.done(function(data, textStatus, jqXHR) {
    // Show progress.
    responseTextArea.val("Handwritten image submitted.");

    // Note: The response may not be immediately available. Handwriting Recognition is an
    // async operation that can take a variable amount of time depending on the length
    // of the text you want to recognize. You may need to wait or retry this GET operation.
    //
    // Try once per second for up to ten seconds to receive the result.
    var tries = 10;
    var waitTime = 100;
    var taskCompleted = false;

    var timeoutID = setInterval(function () {
        // Limit the number of calls.
        if (--tries <= 0) {
            window.clearTimeout(timeoutID);
            responseTextArea.val("The response was not available in the time allowed.");
            return;
        }

        // The "Operation-Location" in the response contains the URI to retrieve the recognized text.
        var operationLocation = jqXHR.getResponseHeader("Operation-Location");

        // Perform the second REST API call and get the response.
        $.ajax({
            url: operationLocation,

            // Request headers.
            beforeSend: function(jqXHR){
                jqXHR.setRequestHeader("Content-Type", "application/json");
                jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                    encodeURIComponent($("#subscriptionKeyInput").val()));
            },

            type: "GET",
        })

        .done(function(data) {
            // If the result is not yet available, return.
            if (data.status && (data.status === "NotStarted" || data.status === "Running")) {
                return;
            }

            // Show formatted JSON on webpage.
            responseTextArea.val(JSON.stringify(data, null, 2));

            // Indicate the task is complete and clear the timer.
            taskCompleted = true;
            window.clearTimeout(timeoutID);
        })

        .fail(function(jqXHR, textStatus, errorThrown) {
            // Indicate the task is complete and clear the timer.
            taskCompleted = true;
            window.clearTimeout(timeoutID);

            // Display error message.
            var errorString = (errorThrown === "") ? "Error: " + errorThrown : " (" + jqXHR.status + "): "

```

```

        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + ")";
";
        errorString += (jqXHR.responseText === "") ? "" :
(jQuery.parseJSON(jqXHR.responseText).message) ?
        jQuery.parseJSON(jqXHR.responseText).message :
jQuery.parseJSON(jqXHR.responseText).error.message;
        alert(errorString);
    });
    }, waitTime);
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Put the JSON description into the text area.
    responseTextArea.val(JSON.stringify(jqXHR, null, 2));

    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + ")";
    errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
        jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
}

```

Handwriting Recognition step 3: Run the application

Save the **handwriting.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image of text to read, then click the **Read Image** button to analyze an image and see the result.

Next steps

- [Computer Vision API C# Tutorial](#)
- [Computer Vision API Python Tutorial](#)

Computer Vision API Python Tutorial

4/25/2018 • 2 minutes to read • [Edit Online](#)

This tutorial shows you how to use the Computer Vision API in Python and how to visualize your results using some popular libraries. Use Jupyter to run the tutorial. To learn how to get started with interactive Jupyter notebooks, refer to: [Jupyter Documentation](#).

Opening the Tutorial Notebook in Jupyter

1. Navigate to the [tutorial notebook in GitHub](#).
2. Click on the green button to clone or download the tutorial.
3. Open a command prompt and go to the folder *Cognitive-Vision-Python-master\Jupyter Notebook*.
4. Run the command **jupyter notebook** from the command prompt. This will start Jupyter.
5. In the Jupyter window, click on *Computer Vision API Example.ipynb* to open the tutorial notebook

Running the Tutorial

To use this notebook, you will need a subscription key for the Computer Vision API. Visit the [Subscription page](#) to sign up. On the "Sign in" page, use your Microsoft account to sign in and you will be able to subscribe and get free keys. After completing the sign-up process, paste your key into the variables section of the notebook (reproduced below). Either the primary or the secondary key works. Make sure to enclose the key in quotes to make it a string.

```
# Variables

_url = 'https://westcentralus.api.cognitive.microsoft.com/vision/v1/analyses'
_key = None #Here you have to paste your primary key
_maxNumRetries = 10
```


How to obtain subscription keys

6/15/2018 • 2 minutes to read • [Edit Online](#)

Computer Vision services require special subscription keys. Every call to the Computer Vision API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header.

To sign up for subscription keys, see [Subscriptions](#). It's free to sign up. Pricing for these services is subject to change.

NOTE

Your subscription keys are valid for only one of these [Microsoft Azure Regions](#).

REGION	ADDRESS
West US	westus.api.cognitive.microsoft.com
East US 2	eastus2.api.cognitive.microsoft.com
West Central US	westcentralus.api.cognitive.microsoft.com
West Europe	westeurope.api.cognitive.microsoft.com
Southeast Asia	southeastasia.api.cognitive.microsoft.com

If you sign up using the Computer Vision free trial, your subscription keys are valid for the **westcentral** region (`https://westcentralus.api.cognitive.microsoft.com/`). That is the most common case. However, if you sign-up for Computer Vision with your Microsoft Azure account through the <https://azure.microsoft.com/> website, you specify the region for your trial from the preceding list of regions.

For example, if you sign up for Computer Vision with your Microsoft Azure account and you specify `westus` for your region, you must use the `westus` region for your REST API calls (`https://westus.api.cognitive.microsoft.com/`).

If you forget the region for your subscription key after obtaining your trial key, you can find your region at <https://azure.microsoft.com/try/cognitive-services/my-apis/>.

Related Links:

- [Pricing Options for Microsoft Cognitive APIs](#)

How to call the Computer Vision API

6/15/2018 • 5 minutes to read • [Edit Online](#)

This guide demonstrates how to call Computer Vision API using REST. The samples are written both in C# using the Computer Vision API client library, and as HTTP POST/GET calls. We will focus on:

- How to get "Tags", "Description" and "Categories".
- How to get "Domain-specific" information (celebrities).

Image URL or path to locally stored image.

- Supported input methods: Raw image binary in the form of an application/octet stream or image URL
- Supported image formats: JPEG, PNG, GIF, BMP
- Image file size: Less than 4MB
- Image dimension: Greater than 50 x 50 pixels

In the examples below, the following features are demonstrated:

1. Analyzing an image and getting an array of tags and a description returned.
2. Analyzing an image with a domain-specific model (specifically, "celebrities" model) and getting the corresponding result in JSON return.

Features are broken down on:

- **Option One:** Scoped Analysis - Analyze only a given model
- **Option Two:** Enhanced Analysis - Analyze to provide additional details with [86-categories taxonomy](#)

Every call to the Computer Vision API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header.

To obtain a subscription key, see [How to Obtain Subscription Keys](#).

1. Passing the subscription key through a query string, see below as a Computer Vision API example:

```
https://westus.api.cognitive.microsoft.com/vision/v2.0/analyze?visualFeatures=Description,Tags&subscription-key=<Your subscription key>
```

2. Passing the subscription key can also be specified in the HTTP request header:

```
ocp-apim-subscription-key: <Your subscription key>
```

3. When using the client library, the subscription key is passed in through the constructor of VisionServiceClient:

```
var visionClient = new VisionServiceClient("Your subscriptionKey");
```

The basic way to perform the Computer Vision API call is by uploading an image directly. This is done by sending a "POST" request with application/octet-stream content type together with the data read from the image. For "Tags" and "Description", this upload method will be the same for all the Computer Vision API calls. The only difference will be the query parameters the user specifies.

Here's how to get "Tags" and "Description" for a given image:

Option One: Get list of "Tags" and one "Description"

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.0/analyze?
visualFeatures=Description,Tags&subscription-key=<Your subscription key>
```

```
using Microsoft.ProjectOxford.Vision;
using Microsoft.ProjectOxford.Vision.Contract;
using System.IO;

AnalysisResult analysisResult;
var features = new VisualFeature[] { VisualFeature.Tags, VisualFeature.Description };

using (var fs = new FileStream(@"C:\Vision\Sample.jpg", FileMode.Open))
{
    analysisResult = await visionClient.AnalyzeImageAsync(fs, features);
}
```

Option Two Get list of "Tags" only, or list of "Description" only:

Tags-only:

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.0/tag&subscription-key=<Your subscription key>
var analysisResult = await visionClient.GetTagsAsync("http://contoso.com/example.jpg");
```

Description-only:

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.0/describe&subscription-key=<Your subscription key>
using (var fs = new FileStream(@"C:\Vision\Sample.jpg", FileMode.Open))
{
    analysisResult = await visionClient.DescribeAsync(fs);
}
```

Here is how to get domain-specific analysis (in our case, for celebrities).

Option One: Scoped Analysis - Analyze only a given model

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.0/models/celebrities/analyze
var celebritiesResult = await visionClient.AnalyzeImageInDomainAsync(url, "celebrities");
```

For this option, all other query parameters {visualFeatures, details} are not valid. If you want to see all supported models, use:

```
GET https://westus.api.cognitive.microsoft.com/vision/v2.0/models
var models = await visionClient.ListModelsAsync();
```

Option Two: Enhanced Analysis - Analyze to provide additional details with [86-categories taxonomy](#)

For applications where you want to get generic image analysis in addition to details from one or more domain-specific models, we extend the v1 API with the models query parameter.

```
POST https://westus.api.cognitive.microsoft.com/vision/v2.0/analyze?details=celebrities
```

When this method is invoked, we will call the 86-category classifier first. If any of the categories match that of a known/matching model, a second pass of classifier invocations will occur. For example, if "details=all", or "details" include 'celebrities', we will call the celebrities model after the 86-category classifier is called and the result includes the category person. This will increase latency for users interested in celebrities, compared to Option One.

All v1 query parameters will behave the same in this case. If visualFeatures=categories is not specified, it will be

implicitly enabled.

Here's an example:

```
{
  "tags": [
    {
      "name": "outdoor",
      "score": 0.976
    },
    {
      "name": "bird",
      "score": 0.95
    }
  ],
  "description": {
    "tags": [
      "outdoor",
      "bird"
    ],
    "captions": [
      {
        "text": "partridge in a pear tree",
        "confidence": 0.96
      }
    ]
  }
}
```

FIELD	TYPE	CONTENT
Tags	object	Top-level object for array of tags
tags[].Name	string	Keyword from tags classifier
tags[].Score	number	Confidence score, between 0 and 1.
description	object	Top-level object for a description.
description.tags[]	string	List of tags. If there insufficient confidence in the ability to produce a caption, the tags maybe the only information available to the caller.
description.captions[].text	string	A phrase describing the image.
description.captions[].confidence	number	Confidence for the phrase.

Option One: Scoped Analysis - Analyze only a given model

The output will be an array of tags, an example will be like this example:

```
{
  "result": [
    {
      "name": "golden retriever",
      "score": 0.98
    },
    {
      "name": "Labrador retriever",
      "score": 0.78
    }
  ]
}
```

Option Two: Enhanced Analysis - Analyze to provide additional details with 86-categories taxonomy

For domain-specific models using Option Two (Enhanced Analysis), the categories return type is extended. An example follows:

```
{
  "requestId": "87e44580-925a-49c8-b661-d1c54d1b83b5",
  "metadata": {
    "width": 640,
    "height": 430,
    "format": "Jpeg"
  },
  "result": {
    "celebrities": [
      {
        "name": "Richard Nixon",
        "faceRectangle": {
          "left": 107,
          "top": 98,
          "width": 165,
          "height": 165
        },
        "confidence": 0.9999827
      }
    ]
  }
}
```

The categories field is a list of one or more of the [86-categories](#) in the original taxonomy. Note also that categories ending in an underscore will match that category and its children (for example, people_ as well as people_group, for celebrities model).

FIELD	TYPE	CONTENT
categories	object	Top-level object
categories[].name	string	Name from 86-category taxonomy
categories[].score	number	Confidence score, between 0 and 1
categories[].detail	object?	Optional detail object

Note that if multiple categories match (for example, 86-category classifier returns a score for both people_ and people_young when model=celebrities), the details are attached to the most general level match (people_ in that example.)

These are identical to vision.analyze, with the additional error of NotSupportedModel error (HTTP 400), which may

be returned in both Option One and Option Two scenarios. For Option Two (Enhanced Analysis), if any of the models specified in details are not recognized, the API will return a `NotSupportedModel`, even if one or more of them are valid. Users can call `listModels` to find out what models are supported.

These are the basic functionalities of the Computer Vision API: how you can upload images and retrieve valuable metadata in return.

To use the REST API, go to [Computer Vision API Reference](#).

How to Analyze Videos in Real-time

4/25/2018 • 7 minutes to read • [Edit Online](#)

This guide will demonstrate how to perform near-real-time analysis on frames taken from a live video stream. The basic components in such a system are:

- Acquire frames from a video source
- Select which frames to analyze
- Submit these frames to the API
- Consume each analysis result that is returned from the API call

These samples are written in C# and the code can be found on GitHub here:

<https://github.com/Microsoft/Cognitive-Samples-VideoFrameAnalysis>.

The Approach

There are multiple ways to solve the problem of running near-real-time analysis on video streams. We will start by outlining three approaches in increasing levels of sophistication.

A Simple Approach

The simplest design for a near-real-time analysis system is an infinite loop, where in each iteration we grab a frame, analyze it, and then consume the result:

```
while (true)
{
    Frame f = GrabFrame();
    if (ShouldAnalyze(f))
    {
        AnalysisResult r = await Analyze(f);
        ConsumeResult(r);
    }
}
```

If our analysis consisted of a lightweight client-side algorithm, this approach would be suitable. However, when our analysis is happening in the cloud, the latency involved means that an API call might take several seconds, during which time we are not capturing images, and our thread is essentially doing nothing. Our maximum frame-rate is limited by the latency of the API calls.

Parallelizing API Calls

While a simple single-threaded loop makes sense for a lightweight client-side algorithm, it doesn't fit well with the latency involved in cloud API calls. The solution to this problem is to allow the long-running API calls to execute in parallel with the frame-grabbing. In C#, we could achieve this using Task-based parallelism, for example:

```

while (true)
{
    Frame f = GrabFrame();
    if (ShouldAnalyze(f))
    {
        var t = Task.Run(async () =>
        {
            AnalysisResult r = await Analyze(f);
            ConsumeResult(r);
        })
    }
}

```

This launches each analysis in a separate Task, which can run in the background while we continue grabbing new frames. This avoids blocking the main thread while waiting for an API call to return, however we have lost some of the guarantees that the simple version provided -- multiple API calls might occur in parallel, and the results might get returned in the wrong order. This could also cause multiple threads to enter the ConsumeResult() function simultaneously, which could be dangerous, if the function is not thread-safe. Finally, this simple code does not keep track of the Tasks that get created, so exceptions will silently disappear. Thus, the final ingredient for us to add is a "consumer" thread that will track the analysis tasks, raise exceptions, kill long-running tasks, and ensure the results get consumed in the correct order, one at a time.

A Producer-Consumer Design

In our final "producer-consumer" system, we have a producer thread that looks very similar to our previous infinite loop. However, instead of consuming analysis results as soon as they are available, the producer simply puts the tasks into a queue to keep track of them.

```

// Queue that will contain the API call tasks.
var taskQueue = new BlockingCollection<Task<ResultWrapper>>();

// Producer thread.
while (true)
{
    // Grab a frame.
    Frame f = GrabFrame();

    // Decide whether to analyze the frame.
    if (ShouldAnalyze(f))
    {
        // Start a task that will run in parallel with this thread.
        var analysisTask = Task.Run(async () =>
        {
            // Put the frame, and the result/exception into a wrapper object.
            var output = new ResultWrapper(f);
            try
            {
                output.Analysis = await Analyze(f);
            }
            catch (Exception e)
            {
                output.Exception = e;
            }
            return output;
        })

        // Push the task onto the queue.
        taskQueue.Add(analysisTask);
    }
}

```

We also have a consumer thread, that is taking tasks off the queue, waiting for them to finish, and either displaying

the result or raising the exception that was thrown. By using the queue, we can guarantee that results get consumed one at a time, in the correct order, without limiting the maximum frame-rate of the system.

```
// Consumer thread.
while (true)
{
    // Get the oldest task.
    Task<ResultWrapper> analysisTask = taskQueue.Take();

    // Await until the task is completed.
    var output = await analysisTask;

    // Consume the exception or result.
    if (output.Exception != null)
    {
        throw output.Exception;
    }
    else
    {
        ConsumeResult(output.Analysis);
    }
}
```

Implementing the Solution

Getting Started

To get your app up and running as quickly as possible, we have implemented the system described above, intending it to be flexible enough to implement many scenarios, while being easy to use. To access the code, go to <https://github.com/Microsoft/Cognitive-Samples-VideoFrameAnalysis>.

The library contains the class `FrameGrabber`, which implements the producer-consumer system discussed above to process video frames from a webcam. The user can specify the exact form of the API call, and the class uses events to let the calling code know when a new frame is acquired, or a new analysis result is available.

To illustrate some of the possibilities, there are two sample apps that use the library. The first is a simple console app, and a simplified version of this is reproduced below. It grabs frames from the default webcam, and submits them to the Face API for face detection.

```

using System;
using VideoFrameAnalyzer;
using Microsoft.ProjectOxford.Face;
using Microsoft.ProjectOxford.Face.Contract;

namespace VideoFrameConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create grabber, with analysis type Face[].
            FrameGrabber<Face[]> grabber = new FrameGrabber<Face[]>();

            // Create Face API Client. Insert your Face API key here.
            FaceServiceClient faceClient = new FaceServiceClient("<subscription key>");

            // Set up our Face API call.
            grabber.AnalysisFunction = async frame => return await
            faceClient.DetectAsync(frame.Image.ToMemoryStream(".jpg"));

            // Set up a listener for when we receive a new result from an API call.
            grabber.NewResultAvailable += (s, e) =>
            {
                if (e.Analysis != null)
                    Console.WriteLine("New result received for frame acquired at {0}. {1} faces detected",
            e.Frame.Metadata.Timestamp, e.Analysis.Length);
            };

            // Tell grabber to call the Face API every 3 seconds.
            grabber.TriggerAnalysisOnInterval(TimeSpan.FromMilliseconds(3000));

            // Start running.
            grabber.StartProcessingCameraAsync().Wait();

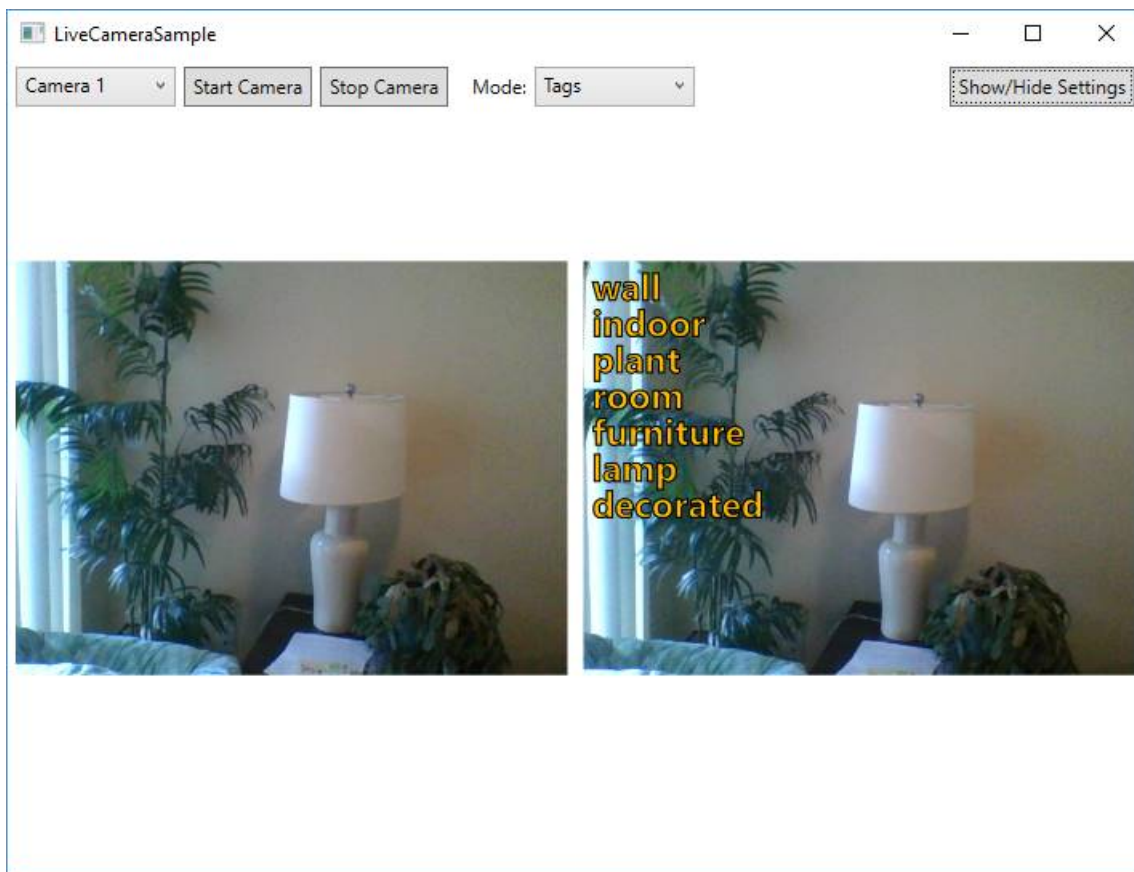
            // Wait for keypress to stop
            Console.WriteLine("Press any key to stop...");
            Console.ReadKey();

            // Stop, blocking until done.
            grabber.StopProcessingAsync().Wait();
        }
    }
}

```

The second sample app is a bit more interesting, and allows you to choose which API to call on the video frames. On the left hand side, the app shows a preview of the live video, on the right hand side it shows the most recent API result overlaid on the corresponding frame.

In most modes, there will be a visible delay between the live video on the left, and the visualized analysis on the right. This delay is the time taken to make the API call. The exception to this is in the "EmotionsWithClientFaceDetect" mode, which performs face detection locally on the client computer using OpenCV, before submitting any images to Cognitive Services. By doing this, we can visualize the detected face immediately, and then update the emotions later once the API call returns. This demonstrates the possibility of a "hybrid" approach, where some simple processing can be performed on the client, and then Cognitive Services APIs can be used to augment this with more advanced analysis when necessary.



Integrating into your codebase

To get started with this sample, follow these steps:

1. Get API keys for the Vision APIs from [Subscriptions](#). For video frame analysis, the applicable APIs are:
 - [Computer Vision API](#)
 - [Emotion API](#)
 - [Face API](#)
2. Clone the [Cognitive-Samples-VideoFrameAnalysis](#) GitHub repo
3. Open the sample in Visual Studio 2015, build and run the sample applications:
 - For BasicConsoleSample, the Face API key is hard-coded directly in [BasicConsoleSample/Program.cs](#).
 - For LiveCameraSample, the keys should be entered into the Settings pane of the app. They will be persisted across sessions as user data.

When you're ready to integrate, **simply reference the VideoFrameAnalyzer library from your own projects.**

Developer Code of Conduct

As with all the Cognitive Services, Developers developing with our APIs and samples are required to follow the "[Developer Code of Conduct for Microsoft Cognitive Services](#)."

The image, voice, video or text understanding capabilities of VideoFrameAnalyzer uses Microsoft Cognitive Services. Microsoft will receive the images, audio, video, and other data that you upload (via this app) and may use them for service improvement purposes. We ask for your help in protecting the people whose data your app sends to Microsoft Cognitive Services.

Summary

In this guide, you learned how to run near-real-time analysis on live video streams using the Face, Computer Vision, and Emotion APIs, and how you can use our sample code to get started. You can get started building your

app with free API keys at the [Microsoft Cognitive Services sign-up page](#).

Please feel free to provide feedback and suggestions in the [GitHub repository](#), or for more broad API feedback, on our [UserVoice site](#).

Connecting to Cognitive Services Computer Vision API by using Connected Services in Visual Studio

6/13/2018 • 6 minutes to read • [Edit Online](#)

By using the Cognitive Services Computer Vision API, you can extract rich information to categorize and process visual data, and perform machine-assisted moderation of images to help curate your services.

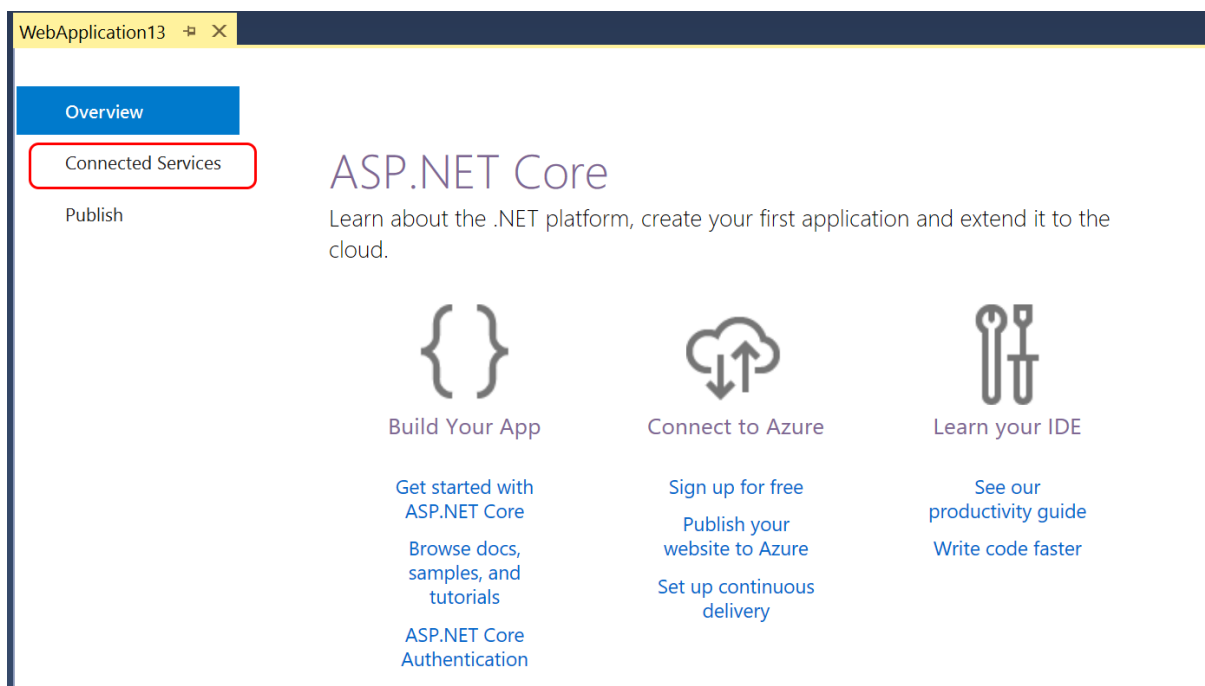
This article and its companion articles provide details for using the Visual Studio Connected Service feature for Cognitive Services Computer Vision API. The capability is available in both Visual Studio 2017 15.7 or later, with the Cognitive Services extension installed.

Prerequisites

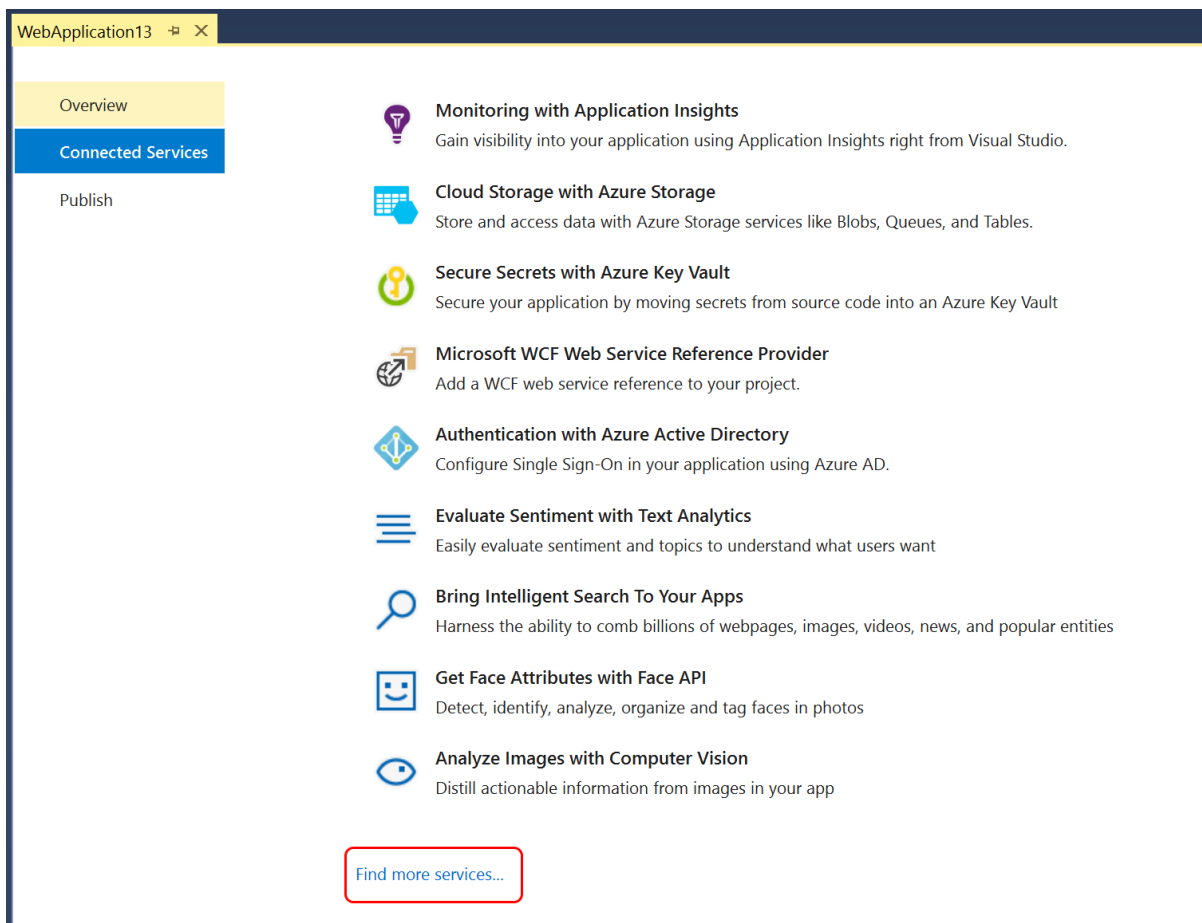
- **An Azure subscription.** If you do not have one, you can sign up for a [free account](#).
- **Visual Studio 2017 version 15.7** with the **Web Development** workload installed. [Download it now](#).

Install the Cognitive Services VSIX Extension

1. With your web project open in Visual Studio, choose the **Connected Services** tab. The tab is available on the welcome page that appears when you open a new project. If you don't see the tab, select **Connected Services** in your project in Solution Explorer.

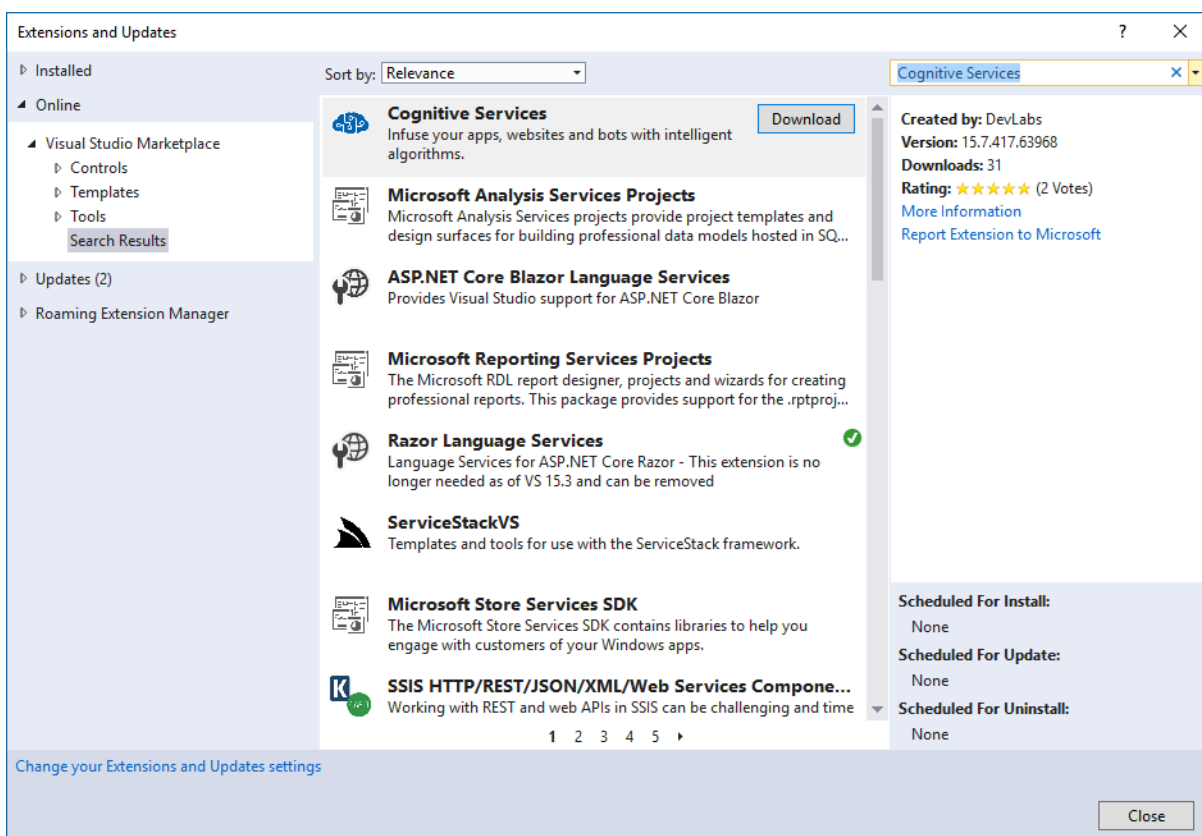


2. Scroll down to the bottom of the list of services, and select **Find more services**.



The **Extensions and Updates** dialog box appears.

3. In the **Extensions and Updates** dialog box, search for **Cognitive Services**, and then download and install the Cognitive Services VSIX package.



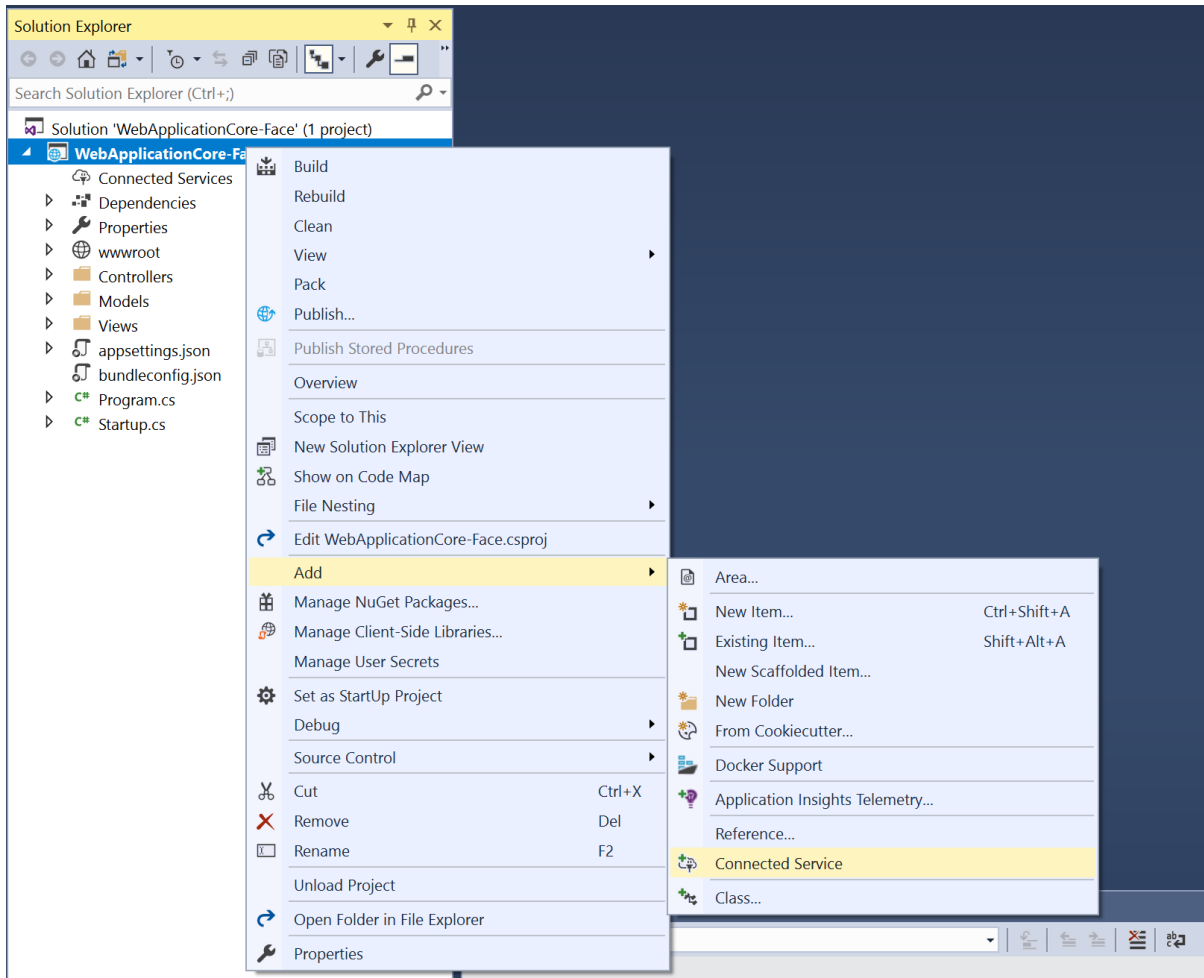
Installing an extension requires a restart of the integrated development environment (IDE).

4. Restart Visual Studio. The extension installs when you close Visual Studio, and is available next time you

launch the IDE.

Add support to your project for Cognitive Services Computer Vision API

1. Create a new ASP.NET Core web project. Use the Empty project template.
2. In **Solution Explorer**, choose **Add > Connected Service**. The Connected Service page appears with services you can add to your project.



3. In the menu of available services, choose **Cognitive Services Computer Vision API**.

WebApplication-Core-ComputerVision

Overview

Connected Services

Add code and dependencies for one of these services to your application

- Monitoring with Application Insights**
Gain visibility into your application using Application Insights right from Visual Studio.
- Cloud Storage with Azure Storage**
Store and access data with Azure Storage services like Blobs, Queues, and Tables.
- Secure Secrets with Azure Key Vault**
Secure your application by moving secrets from source code into an Azure Key Vault
- Microsoft WCF Web Service Reference Provider**
Add a WCF web service reference to your project.
- Analyze Images with Computer Vision**
Distill actionable information from images in your app.
- Evaluate Sentiment with Text Analytics**
Easily evaluate sentiment and topics to understand what users want
- Bring Intelligent Search To Your Apps**
Harness the ability to comb billions of webpages, images, videos, news, and popular entities.
- Get Face Attributes with Face API**
Detect, identify, analyze, organize and tag faces in photos
- Translate Content with Speech Service**
Convert spoken audio into text, use voice for verification, or add speaker recognition to your app.

If you've signed into Visual Studio, and have an Azure subscription associated with your account, a page appears with a dropdown list with your subscriptions.

WebApplicationCor...omputer Vision API

WebApplicationCore-Cog1

Computer Vision API

Distill actionable information from images in your app.

Subscription: Microsoft Azure Internal Consumption

Name: WebApplicationCore-Cog1_ComputerVisionAPI (ne

Edit...

Adding an Azure Computer Vision API will:

- Create a new Computer Vision API in resource group 'WebApplicationCore-Cog1_rg' in Australia East.
- Create a new resource group to host your Computer Vision API
- Use the pricing tier F0 - Free
- <What you can do with a cognitive service>

[More About Computer Vision API](#)

[Review pricing](#)

Add

4. Select the subscription you want to use, and then choose a name for the Computer Vision API, or choose the Edit link to modify the automatically generated name, choose the resource group, and the Pricing Tier.

✕

Edit Azure Computer Vision API

Name:

Resource Group:

Location:

Pricing tier:

[Review pricing](#)

Follow the link for details on the pricing tiers.

5. Choose Add to add supported for the Connected Service. Visual Studio modifies your project to add the NuGet packages, configuration file entries, and other changes to support a connection the Computer Vision API. The Output Window shows the log of what is happening to your project. You should see something like the following:

```
[4/26/2018 5:15:31.664 PM] Adding Computer Vision API to the project.
[4/26/2018 5:15:32.084 PM] Creating new ComputerVision...
[4/26/2018 5:15:32.153 PM] Creating new Resource Group...
[4/26/2018 5:15:40.286 PM] Installing NuGet package
'Microsoft.Azure.CognitiveServices.Vision.ComputerVision' version 1.0.2-preview.
[4/26/2018 5:15:44.117 PM] Retrieving keys...
[4/26/2018 5:15:45.602 PM] Changing appsettings.json setting: ComputerVisionAPI_ServiceKey=<service key>
[4/26/2018 5:15:45.606 PM] Changing appsettings.json setting:
ComputerVisionAPI_ServiceEndPoint=https://australiaeast.api.cognitive.microsoft.com/vision/v2.0
[4/26/2018 5:15:45.609 PM] Changing appsettings.json setting: ComputerVisionAPI_Name=WebApplication-
Core-ComputerVision_ComputerVisionAPI
[4/26/2018 5:15:46.747 PM] Successfully added Computer Vision API to the project.
```

Use the Computer Vision API to detect attributes of an image

1. Add the following using statements in Startup.cs.

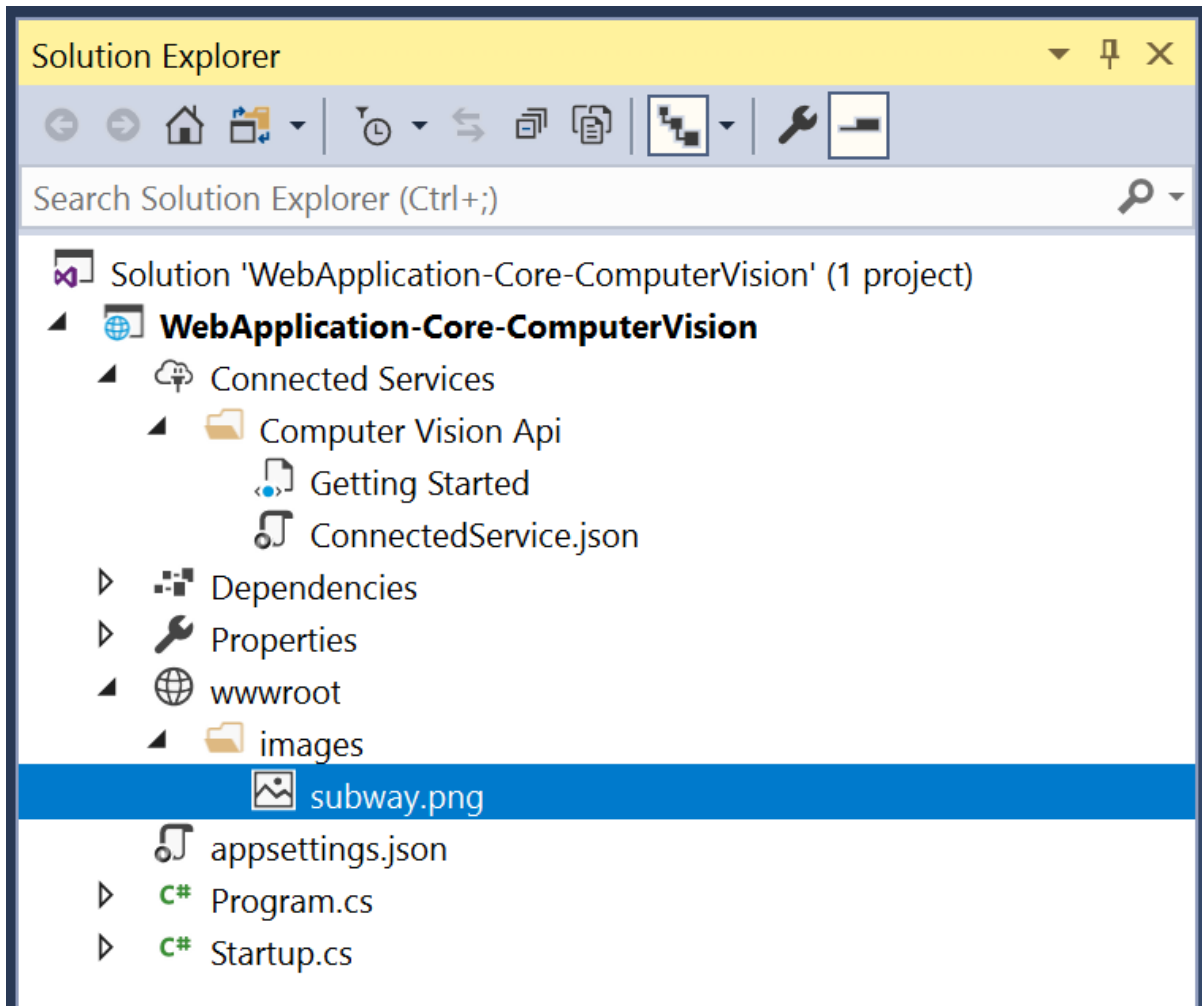
```
using System.IO;
using System.Text;
using Microsoft.Extensions.Configuration;
using System.Net.Http;
using System.Net.Http.Headers;
```

2. Add a configuration field, and add a constructor that initializes the configuration field in the `Startup` class to enable configuration in your program.

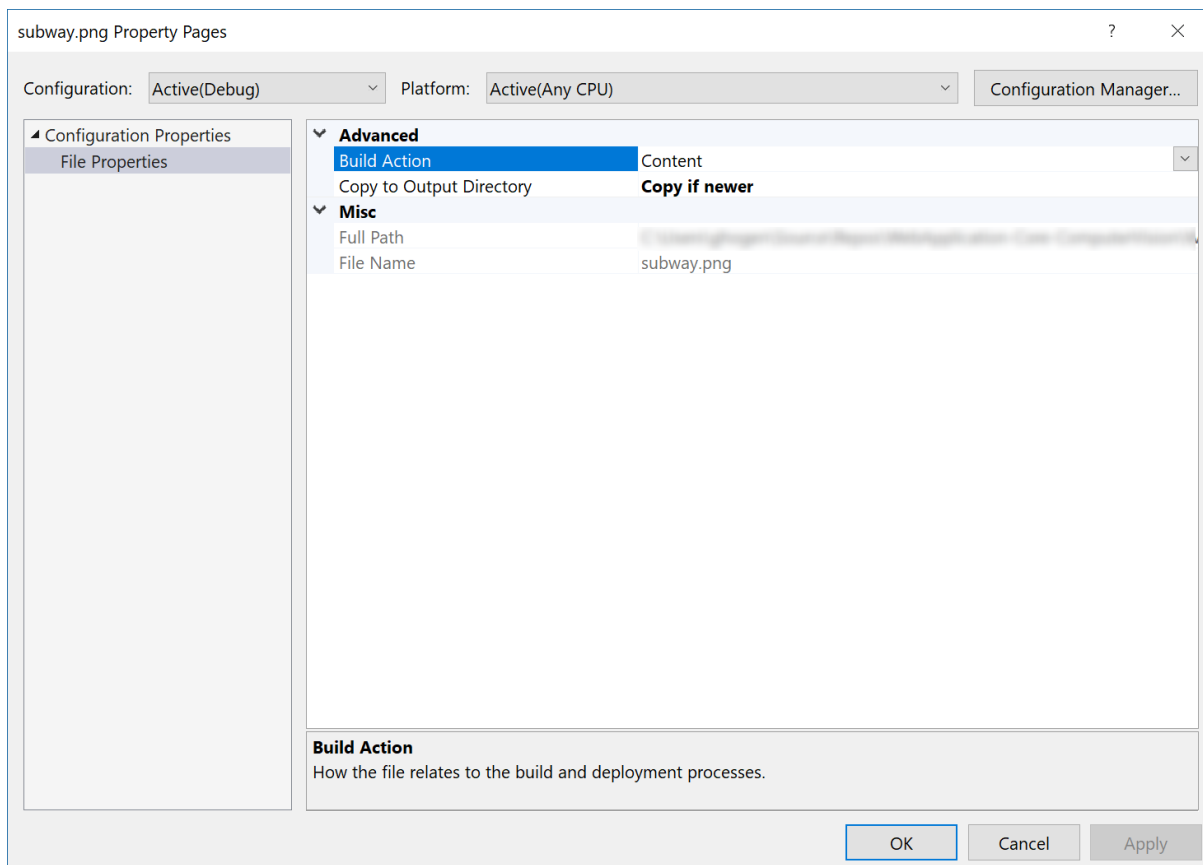
```
private IConfiguration configuration;

public Startup(IConfiguration configuration)
{
    this.configuration = configuration;
}
```

3. In the wwwroot folder in your project, add an images folder, and add an image file to your wwwroot folder. As an example, you can use one of the images on this [Computer Vision API page](#). Right-click on one of the images, save to your local hard drive, then in Solution Explorer, right-click on the images folder, and choose **Add > Existing Item** to add it to your project. Your project should look something like this in Solution Explorer:



4. Right-click on the image file, choose Properties, and then choose **Copy if newer**.



5. Replace the Configure method with the following code to access the Computer Vision API and test an image.

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    // TODO: Change this to your image's path on your site.
    string imagePath = @"images/subway.png";

    // Enable static files such as image files.
    app.UseStaticFiles();

    string visionApiKey = this.configuration["ComputerVisionAPI_ServiceKey"];
    string visionApiEndPoint = this.configuration["ComputerVisionAPI_ServiceEndPoint"];

    HttpClient client = new HttpClient();

    // Request headers.
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", visionApiKey);

    // Request parameters. A third optional parameter is "details".
    string requestParameters = "visualFeatures=Categories,Description,Color&language=en";

    // Assemble the URI for the REST API Call.
    string uri = visionApiEndPoint + "/analyze" + "?" + requestParameters;

    HttpResponseMessage response;

    // Request body. Posts an image you've added to your site's images folder.
    var fileInfo = env.WebRootFileProvider.GetFileInfo(imagePath);
    byte[] byteData = GetImageAsByteArray(fileInfo.PhysicalPath);

    string contentString = string.Empty;
    using (ByteArrayContent content = new ByteArrayContent(byteData))
    {
        // This example uses content type "application/octet-stream".
        // The other content types you can use are "application/json" and "multipart/form-data".
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

        // Execute the REST API call.
        response = client.PostAsync(uri, content).Result;

        // Get the JSON response.
        contentString = response.Content.ReadAsStringAsync().Result;
    }

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("<h1>Cognitive Services Demo</h1>");
        await context.Response.WriteAsync("<p><b>Test Image:</b></p>");
        await context.Response.WriteAsync("<div><img src=\"\" + imagePath + \"\" /></div>");
        await context.Response.WriteAsync("<p><b>Computer Vision API results:</b></p>");
        await context.Response.WriteAsync("<p>");
        await context.Response.WriteAsync(JsonPrettyPrint(contentString));
        await context.Response.WriteAsync("<p>");
    });
}
```

The code here constructs a HTTP request with the URI and the image as binary content for a call to the Computer Vision REST API.

6. Add the helper functions GetImageAsByteArray and JsonPrettyPrint.

```
/// <summary>
```

```

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}

/// <summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "    ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {
            case '{':
            case '[':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    indent++;
                }
                break;
            case '}':
            case ']':
                if (!quoted)
                {
                    sb.AppendLine();
                    indent--;
                }
                sb.Append(ch);
                break;
            case '"':
                sb.Append(ch);
                bool escaped = false;
                var index = i;
                while (index > 0 && json[--index] == '\\')
                    escaped = !escaped;
                if (!escaped)
                    quoted = !quoted;
                break;
            case ',':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    indent--;
                }
                break;
            case ':':
                sb.Append(ch);
                if (!quoted)
                    sb.Append(" ");
                break;
        }
    }
    return sb.ToString();
}

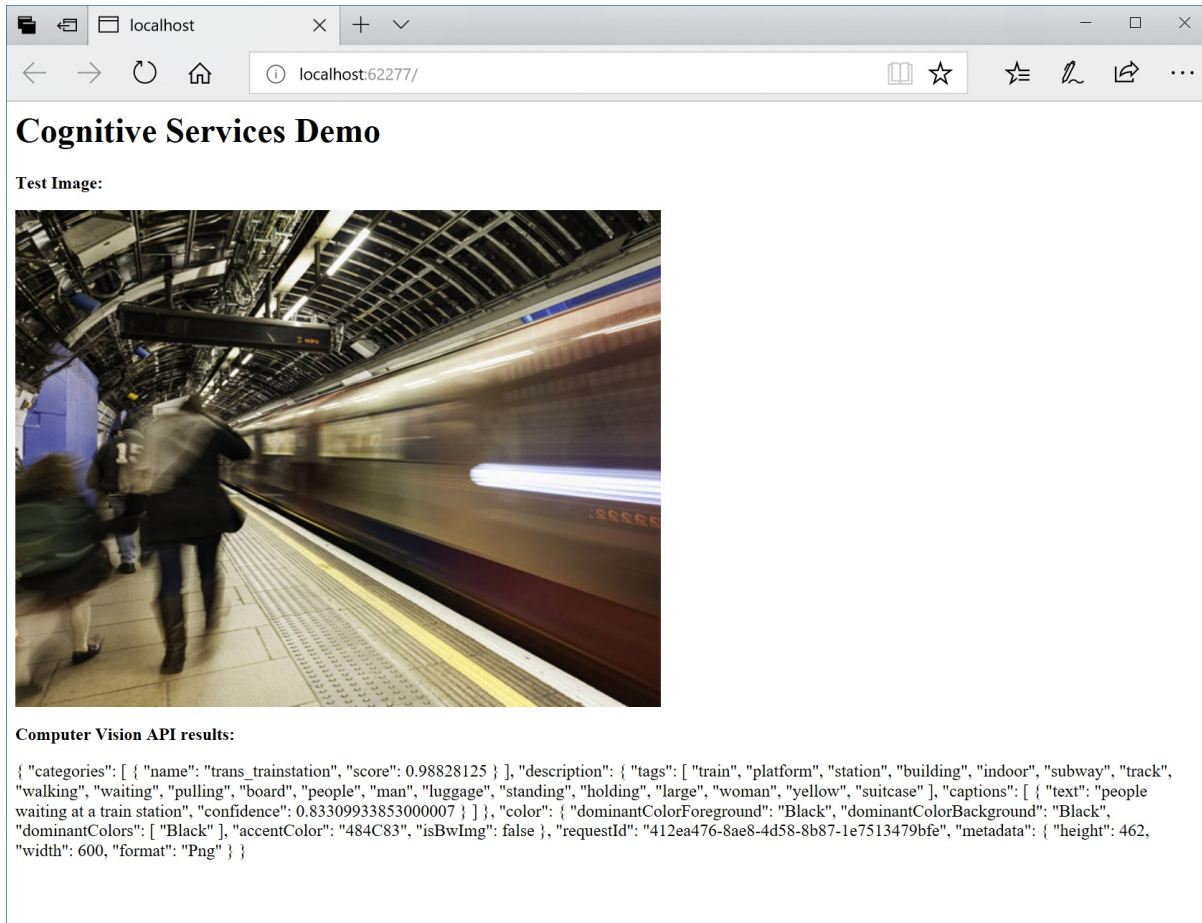
```

```

        break;
    default:
        sb.Append(ch);
        break;
    }
}
return sb.ToString();
}


```

7. Run the web application and see what Computer Vision API found in your image.



Cognitive Services Demo

Test Image:



Computer Vision API results:

```

{ "categories": [ { "name": "trans_trainstation", "score": 0.98828125 }, { "description": { "tags": [ "train", "platform", "station", "building", "indoor", "subway", "track", "walking", "waiting", "pulling", "board", "people", "man", "luggage", "standing", "holding", "large", "woman", "yellow", "suitcase" ], "captions": [ { "text": "people waiting at a train station", "confidence": 0.83309933853000007 } ] }, "color": { "dominantColorForeground": "Black", "dominantColorBackground": "Black", "dominantColors": [ "Black" ], "accentColor": "484C83", "isBwImg": false }, "requestId": "412ea476-8ae8-4d58-8b87-1e7513479bfe", "metadata": { "height": 462, "width": 600, "format": "Png" } } }

```

Clean up resources

When no longer needed, delete the resource group. This deletes the cognitive service and related resources. To delete the resource group through the portal:

1. Enter the name of your resource group in the Search box at the top of the portal. When you see the resource group used in this QuickStart in the search results, select it.
2. Select **Delete resource group**.
3. In the **TYPE THE RESOURCE GROUP NAME:** box type in the name of the resource group and select **Delete**.

Next steps

Learn more about the Computer Vision API by reading the [Computer Vision API Documentation](#).

Computer Vision API Frequently Asked Questions

4/19/2018 • 2 minutes to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, try asking the Computer Vision API community on [StackOverflow](#) or contact [Help and Support on UserVoice](#)

Question: *Can I train Computer Vision API to use custom tags? For example, I would like to feed in pictures of cat breeds to 'train' the AI, then receive the breed value on an AI request.*

Answer: This function is currently not available. However, our engineers are working to bring this functionality to Computer Vision.

Question: *Can Computer Vision be used locally without an internet connection?*

Answer: We currently do not offer an on-premises or local solution.

Question: *Which languages are supported with Computer Vision?*

Answer: Supported languages include:

		SUPPORTED LANGUAGES		
Danish (da-DK)	Dutch (nl-NL)	English	Finnish (fi-FI)	French (fr-FR)
German (de-DE)	Greek (el-GR)	Hungarian (hu-HU)	Italian (it-IT)	Japanese (ja-JP)
Korean (ko-KR)	Norwegian (nb-NO)	Polish (pl-PL)	Portuguese (pt-BR) (pt-PT)	Russian (ru-RU)
Spanish (es-ES)	Swedish (sv-SV)	Turkish (tr-TU)		

Question: *Can Computer Vision be used to read license plates?*

Answer: The Vision API offers good text-detection with OCR, but it is not currently optimized for license plates. We are constantly trying to improve our services and have added OCR for auto license plate recognition to our list of feature requests.

Question: *Which languages are supported for handwriting recognition?*

Answer: Currently, only English is supported.

Question: *What types of writing surfaces are supported for handwriting recognition?*

Answer: The technology works with different kinds of surfaces, including whiteboards, white paper, and yellow sticky notes.

Question: *How long does the handwriting recognition operation take?*

Answer: The amount of time that it takes depends on the length of the text. For longer texts, it can take up to several seconds. Therefore, after the Recognize Handwritten Text operation completes, you may need to wait before you can retrieve the results using the Get Handwritten Text Operation Result operation.

Question: *How does the handwriting recognition technology handle text that was inserted using a caret in the middle of a line?*

Answer: Such text is returned as a separate line by the handwriting recognition operation.

Question: *How does the handwriting recognition technology handle crossed-out words or lines?*

Answer: If the words are crossed out with multiple lines to render them unrecognizable, the handwriting recognition operation doesn't pick them up. However, if the words are crossed out using a single line, that crossing is treated as noise, and the words still get picked up by the handwriting recognition operation.

Question: *What text orientations are supported for the handwriting recognition technology?*

Answer: Text oriented at angles of up to around 30 degrees to 40 degrees may get picked up by the handwriting recognition operation.

86-Categories Taxonomy

4/19/2018 • 2 minutes to read • [Edit Online](#)

abstract_

abstract_net

abstract_nonphoto

abstract_rect

abstract_shape

abstract_texture

animal_

animal_bird

animal_cat

animal_dog

animal_horse

animal_panda

building_

building_arch

building_brickwall

building_church

building_corner

building_doorwindows

building_pillar

building_stair

building_street

dark_

drink_

drink_can

dark_fire

dark_fireworks

sky_object

food_

food_bread

food_fastfood

food_grilled
food_pizza
indoor_
indoor_churchwindow
indoor_court
indoor_doorwindows
indoor_marketstore
indoor_room
indoor_venue
dark_light
others_
outdoor_
outdoor_city
outdoor_field
outdoor_grass
outdoor_house
outdoor_mountain
outdoor_oceanbeach
outdoor_playground
outdoor_railway
outdoor_road
outdoor_sportsfield
outdoor_stonerock
outdoor_street
outdoor_water
outdoor_waterside
people_
people_baby
people_crowd
people_group
people_hand
people_many
people_portrait

people_show

people_tattoo

people_young

plant_

plant_branch

plant_flower

plant_leaves

plant_tree

object_screen

object_sculpture

sky_cloud

sky_sun

people_swimming

outdoor_pool

text_

text_mag

text_map

text_menu

text_sign

trans_bicycle

trans_bus

trans_car

trans_trainstation