Quickstart: Use the Computer Vision client library

06/29/2020 • 53 minutes to read •

Choose a programming language

C#	Go	Java	JavaScript	Python

In this article

Prerequisites
Setting up
Object model
Code examples
Authenticate the client
Analyze an image
Read printed and handwritten text
Run the application
Clean up resources
Next steps

Get started with the Computer Vision client library. Follow these steps to install the package and try out the example code for basic tasks. Computer Vision provides you with access to advanced algorithms for processing images and returning information.

Use the Computer Vision client library to:

- Analyze an image for tags, text description, faces, adult content, and more.
- Recognize printed and handwritten text with the Batch Read API.

Reference documentation | Library source code | Package (npm) | Samples

Prerequisites

- An Azure subscription - Create one for free
- The current version of Node.js
- Once you have your Azure subscription, create a Computer Vision resource in the Azure portal to get your key and endpoint. After it deploys, click **Go to resource**.

- You will need the key and endpoint from the resource you create to connect your application to the Computer Vision service. You'll paste your key and endpoint into the code below later in the quickstart.
- You can use the free pricing tier (`F0`) to try the service, and upgrade later to a paid tier for production.

- Create environment variables for the key and endpoint URL, named `COMPUTER_VISION_SUBSCRIPTION_KEY` and `COMPUTER_VISION_ENDPOINT`, respectively.

Setting up

Create a new Node.js application

In a console window (such as cmd, PowerShell, or Bash), create a new directory for your app, and navigate to it.

Console Copy

```
mkdir myapp && cd myapp
```

Run the `npm init` command to create a node application with a `package.json` file.

Console Copy

```
npm init
```

Install the client library

Install the `ms-rest-azure` and `@azure/cognitiveservices-computervision` NPM packages:

Console Copy

```
npm install @azure/cognitiveservices-computervision
```

Your app's `package.json` file will be updated with the dependencies.

Prepare the Node.js script

Create a new file, *index.js*, and open it in a text editor. Add the following import statements.

JavaScript Copy

```
'use strict';

const async = require('async');
const fs = require('fs');
const path = require("path");
const createReadStream = require('fs').createReadStream
const sleep = require('util').promisify(setTimeout);
const ComputerVisionClient = require('@azure/cognitiveservices-
computervision').ComputerVisionClient;
const ApiKeyCredentials = require('@azure/ms-rest-js').ApiKeyCredentials;
```

Then, define a function `computerVision` and declare an async series with primary function and callback function. You will add your quickstart code into the primary function, and call `computerVision` at the bottom of the script.

JavaScript Copy

```
function computerVision() {
  async.series([
    async function () {
```

JavaScript Copy

```
    },
    function () {
      return new Promise((resolve) => {
        resolve();
      })
    }
  ], (err) => {
    throw (err);
  });
}

computerVision();
```

Object model

The following classes and interfaces handle some of the major features of the Computer Vision Node.js SDK.

Name	Description

Name	Description
ComputerVisionClient	This class is needed for all Computer Vision functionality. You instantiate it with your subscription information, and you use it to do most image operations.
VisualFeatureTypes	This enum defines the different types of image analysis that can be done in a standard Analyze operation. You specify a set of **VisualFeatureTypes** values depending on your needs.

Code examples

These code snippets show you how to do the following tasks with the Computer Vision client library for Node.js:

- Authenticate the client
- Analyze an image
- Read printed and handwritten text

Authenticate the client

Create variables for your resource's Azure endpoint and key. If you created the environment variable after you launched the application, you will need to close and reopen the editor, IDE, or shell running it to access the variable.

JavaScript Copy

```javascript
/**
 * AUTHENTICATE
 * This single client is used for all examples.
 */
let key = process.env['COMPUTER_VISION_SUBSCRIPTION_KEY'];
let endpoint = process.env['COMPUTER_VISION_ENDPOINT']
if (!key) { throw new Error('Set your environment variables for your
subscription key and endpoint.'); }
```

Instantiate a client with your endpoint and key. Create a ApiKeyCredentials object with your key and endpoint, and use it to create a ComputerVisionClient object.

JavaScript Copy

```javascript
let computerVisionClient = new ComputerVisionClient(
    new ApiKeyCredentials({inHeader: {'Ocp-Apim-Subscription-Key': key}}),
endpoint);
```

Analyze an image

The code in this section analyzes remote images to extract various visual features. You can do these operations as part of the **analyzeImage** method of the client object, or you can call them using individual methods. See the reference documentation for details.

> **Note**
>
> You can also analyze a local image. See the sample code on **GitHub** for scenarios involving local images.

Get image description

The following code gets the list of generated captions for the image. See Describe images for more details.

First, define the URL of an image to analyze:

JavaScript Copy

```javascript
var describeURL =
'https://moderatorsampleimages.blob.core.windows.net/samples/sample1.jpg';
```

Then add the following code to get the image description and print it to the console.

JavaScript Copy

```javascript
// Analyze URL image
console.log('Analyzing URL image to describe...',
describeURL.split('/').pop());
var caption = (await
computerVisionClient.describeImage(describeURL)).captions[0];
console.log(`This may be ${caption.text} (${caption.confidence.toFixed(2)}
confidence)`);
```

Get image category

The following code gets the detected category of the image. See Categorize images for more details.

JavaScript Copy

```
const categoryURLImage =
'https://moderatorsampleimages.blob.core.windows.net/samples/sample16.png';

// Analyze URL image
console.log('Analyzing category in image...',
categoryURLImage.split('/').pop());
let categories = (await
computerVisionClient.analyzeImage(categoryURLImage)).categories;
console.log(`Categories: ${formatCategories(categories)}`);
```

Define the helper function `formatCategories`:

JavaScript Copy

```
// Formats the image categories
function formatCategories(categories) {
  categories.sort((a, b) => b.score - a.score);
  return categories.map(cat => `${cat.name}
(${cat.score.toFixed(2)})`).join(', ');
}
```

Get image tags

The following code gets the set of detected tags in the image. See Content tags for more details.

JavaScript Copy

```
console.log('-------------------------------------------------');
console.log('DETECT TAGS');
console.log();

// Image of different kind of dog.
const tagsURL =
'https://moderatorsampleimages.blob.core.windows.net/samples/sample16.png';

// Analyze URL image
console.log('Analyzing tags in image...', tagsURL.split('/').pop());
let tags = (await computerVisionClient.analyzeImage(tagsURL,
{visualFeatures: ['Tags']})).tags;
console.log(`Tags: ${formatTags(tags)}`);
```

Define the helper function `formatTags`:

JavaScript Copy

```
// Format tags for display
function formatTags(tags) {
  return tags.map(tag => (`${tag.name}
(${tag.confidence.toFixed(2)})`)).join(', ');
}
```

Detect objects

The following code detects common objects in the image and prints them to the console. See Object detection for more details.

JavaScript Copy

```
// Image of a dog
const objectURL = 'https://raw.githubusercontent.com/Azure-
Samples/cognitive-services-node-sdk-samples/master/Data/image.jpg';

// Analyze a URL image
console.log('Analyzing objects in image...', objectURL.split('/').pop());
let objects = (await computerVisionClient.analyzeImage(objectURL,
{visualFeatures: ['Objects']})).objects;
console.log();

// Print objects bounding box and confidence
if (objects.length) {
    console.log(`${objects.length} object${objects.length == 1 ? '' : 's'}
found:`);
    for (let obj of objects) { console.log(`    ${obj.object}
(${obj.confidence.toFixed(2)}) at ${formatRectObjects(obj.rectangle)}`); }
} else { console.log('No objects found.'); }
```

Define the helper function `formatRectObjects`:

JavaScript Copy

```
// Formats the bounding box
function formatRectObjects(rect) {
  return `top=${rect.y}`.padEnd(10) + `left=${rect.x}`.padEnd(10) +
`bottom=${rect.y + rect.h}`.padEnd(12)
  + `right=${rect.x + rect.w}`.padEnd(10) + `(${rect.w}x${rect.h})`;
}
```

Detect brands

The following code detects corporate brands and logos in the image and prints them to the console. See Brand detection for more details.

JavaScript Copy

```javascript
const brandURLImage = 'https://docs.microsoft.com/en-us/azure/cognitive-
services/computer-vision/images/red-shirt-logo.jpg';

// Analyze URL image
console.log('Analyzing brands in image...', brandURLImage.split('/').pop());
let brands = (await computerVisionClient.analyzeImage(brandURLImage,
{visualFeatures: ['Brands']})).brands;

// Print the brands found
if (brands.length) {
    console.log(`${brands.length} brand${brands.length != 1 ? 's' : ''}
found:`);
    for (let brand of brands) {
        console.log(`    ${brand.name} (${brand.confidence.toFixed(2)}
confidence)`);
    }
} else { console.log(`No brands found.`); }
```

Detect faces

The following code returns the detected faces in the image with their rectangle coordinates and select face attributes. See Face detection for more details.

JavaScript Copy

```javascript
const facesImageURL = 'https://raw.githubusercontent.com/Azure-
Samples/cognitive-services-sample-data-
files/master/ComputerVision/Images/faces.jpg';

// Analyze URL image.
console.log('Analyzing faces in image...', facesImageURL.split('/').pop());
// Get the visual feature for 'Faces' only.
let faces = (await computerVisionClient.analyzeImage(facesImageURL,
{visualFeatures: ['Faces']})).faces;

// Print the bounding box, gender, and age from the faces.
if (faces.length) {
  console.log(`${faces.length} face${faces.length == 1 ? '' : 's'} found:`);
  for (let face of faces) { console.log(`    Gender:
${face.gender}`.padEnd(20)
    + ` Age: ${face.age}`.padEnd(10) + `at
${formatRectFaces(face.faceRectangle)}`); }
} else { console.log('No faces found.'); }
```

Define the helper function `formatRectFaces`:

JavaScript Copy

```
// Formats the bounding box
function formatRectFaces(rect) {
  return `top=${rect.top}`.padEnd(10) + `left=${rect.left}`.padEnd(10) +
`bottom=${rect.top + rect.height}`.padEnd(12)
    + `right=${rect.left + rect.width}`.padEnd(10) +
`(${rect.width}x${rect.height})`;
}
```

Detect adult, racy, or gory content

The following code prints the detected presence of adult content in the image. See Adult, racy, gory content for more details.

Define the URL of the image to use:

JavaScript Copy

```
// The URL image and local images are not racy/adult.
// Try your own racy/adult images for a more effective result.
const adultURLImage = 'https://raw.githubusercontent.com/Azure-
Samples/cognitive-services-sample-data-
files/master/ComputerVision/Images/celebrities.jpg';
```

Then add the following code to detect adult content and print the results to the console.

JavaScript Copy

```
// Function to confirm racy or not
const isIt = flag => flag ? 'is' : "isn't";

// Analyze URL image
console.log('Analyzing image for racy/adult content...',
adultURLImage.split('/').pop());
var adult = (await computerVisionClient.analyzeImage(adultURLImage, {
  visualFeatures: ['Adult']
})).adult;
console.log(`This probably ${isIt(adult.isAdultContent)} adult content
(${adult.adultScore.toFixed(4)} score)`);
console.log(`This probably ${isIt(adult.isRacyContent)} racy content
(${adult.racyScore.toFixed(4)} score)`);
```

Get image color scheme

The following code prints the detected color attributes in the image, like the dominant colors and accent color. See Color schemes for more details.

JavaScript Copy

```
const colorURLImage = 'https://raw.githubusercontent.com/Azure-
Samples/cognitive-services-sample-data-
files/master/ComputerVision/Images/celebrities.jpg';

// Analyze URL image
console.log('Analyzing image for color scheme...',
colorURLImage.split('/').pop());
console.log();
let color = (await computerVisionClient.analyzeImage(colorURLImage,
{visualFeatures: ['Color']})).color;
printColorScheme(color);
```

Define the helper function `printColorScheme` to print the details of the color scheme to the console.

JavaScript Copy

```
// Print a detected color scheme
function printColorScheme(colors){
  console.log(`Image is in ${colors.isBwImg ? 'black and white' :
'color'}`);
  console.log(`Dominant colors: ${colors.dominantColors.join(', ')}`);
  console.log(`Dominant foreground color:
${colors.dominantColorForeground}`);
  console.log(`Dominant background color:
${colors.dominantColorBackground}`);
  console.log(`Suggested accent color: #${colors.accentColor}`);
}
```

Get domain-specific content

Computer Vision can use specialized model to do further analysis on images. See Domain-specific content for more details.

First, define the URL of an image to analyze:

JavaScript Copy

```
const domainURLImage = 'https://raw.githubusercontent.com/Azure-
Samples/cognitive-services-sample-data-
files/master/ComputerVision/Images/landmark.jpg';
```

The following code parses data about detected landmarks in the image.

JavaScript Copy

```
// Analyze URL image
console.log('Analyzing image for landmarks...',
domainURLImage.split('/').pop());
let domain = (await computerVisionClient.analyzeImageByDomain('landmarks',
domainURLImage)).result.landmarks;

// Prints domain-specific, recognized objects
if (domain.length) {
  console.log(`${domain.length} ${domain.length == 1 ? 'landmark' :
'landmarks'} found:`);
  for (let obj of domain) {
    console.log(`    ${obj.name}`.padEnd(20) +
`(${obj.confidence.toFixed(2)} confidence)`.padEnd(20) +
`${formatRectDomain(obj.faceRectangle)}`);
  }
} else {
  console.log('No landmarks found.');
}
```

Define the helper function `formatRectDomain` to parse the location data about detected landmarks.

JavaScript Copy

```javascript
// Formats bounding box
function formatRectDomain(rect) {
  if (!rect) return '';
  return `top=${rect.top}`.padEnd(10) + `left=${rect.left}`.padEnd(10) +
`bottom=${rect.top + rect.height}`.padEnd(12) +
    `right=${rect.left + rect.width}`.padEnd(10) +
`(${rect.width}x${rect.height})`;
}
```

Get the image type

The following code prints information about the type of image—whether it is clip art or line drawing.

JavaScript Copy

```javascript
const typeURLImage = 'https://raw.githubusercontent.com/Azure-
Samples/cognitive-services-python-sdk-
samples/master/samples/vision/images/make_things_happen.jpg';

 // Analyze URL image
console.log('Analyzing type in image...', typeURLImage.split('/').pop());
let types = (await computerVisionClient.analyzeImage(typeURLImage,
{visualFeatures: ['ImageType']})).imageType;
console.log(`Image appears to be ${describeType(types)}`);
```

Define the helper function `describeType`:

JavaScript Copy

```javascript
function describeType(imageType) {
  if (imageType.clipArtType && imageType.clipArtType >
imageType.lineDrawingType) return 'clip art';
  if (imageType.lineDrawingType && imageType.clipArtType <
imageType.lineDrawingType) return 'a line drawing';
  return 'a photograph';
}
```

Read printed and handwritten text

Computer Vision can read visible text in an image and convert it to a character stream.

Note

You can also read text from a local image. See the sample code on **GitHub** for scenarios involving local images.

Set up test images

Save a reference of the URL of the images you want to extract text from.

JavaScript Copy

```javascript
// URL images containing printed and handwritten text
 const printedText     =
'https://moderatorsampleimages.blob.core.windows.net/samples/sample2.jpg';
 const handwrittenText =
'https://raw.githubusercontent.com/MicrosoftDocs/azure-
docs/master/articles/cognitive-services/Computer-
vision/Images/readsample.jpg';
```

Call the Recognize API

Add the code below, which calls the `recognizeText` function for the given images.

JavaScript Copy

```
// Recognize text in printed image
console.log('Recognizing printed text...', printedText.split('/').pop());
var printed = await recognizeText(computerVisionClient, 'Printed',
printedText);
printRecText(printed);

// Recognize text in handwritten image
console.log('\nRecognizing handwritten text...',
handwrittenText.split('/').pop());
var handwriting = await recognizeText(computerVisionClient, 'Handwritten',
handwrittenText);
printRecText(handwriting);
```

Define the `recognizeText` function. This calls the **recognizeText** method on the client object, which returns an operation ID and starts an asynchronous process to read the content of the image. Then it uses the operation ID to check the operation at one-second intervals until the results are returned. It then returns the extracted results.

JavaScript Copy

```
// Perform text recognition and await the result
async function recognizeText(client, mode, url) {
  // To recognize text in a local image, replace client.recognizeText() with
recognizeTextInStream() as shown:
  // result = await client.recognizeTextInStream(mode, () =>
createReadStream(localImagePath));
  let result = await client.recognizeText(mode, url);
  // Operation ID is last path segment of operationLocation (a URL)
  let operation = result.operationLocation.split('/').slice(-1)[0];

  // Wait for text recognition to complete
  // result.status is initially undefined, since it's the result of
recognizeText
  while (result.status !== 'Succeeded') { await sleep(1000); result = await
client.getTextOperationResult(operation); }
  return result.recognitionResult;
}
```

Then, define the helper function `printRecText`, which prints the results of a Recognize operation to the console.

JavaScript Copy

```
// Prints all text from OCR result
function printRecText(ocr) {
  if (ocr.lines.length) {
      console.log('Recognized text:');
      for (let line of ocr.lines) {
          console.log(line.words.map(w => w.text).join(' '));
```

```
        }
    }
    else { console.log('No recognized text.'); }
}
```

Run the application

Run the application with the `node` command on your quickstart file.

Console Copy

```
node index.js
```

Clean up resources

If you want to clean up and remove a Cognitive Services subscription, you can delete the resource or resource group. Deleting the resource group also deletes any other resources associated with it.

- Portal
- Azure CLI

Next steps

Computer Vision API reference (Node.js)

- What is Computer Vision?
- The source code for this sample can be found on GitHub.

Is this page helpful?

Yes No