

# Table of Contents

[Overview](#)

[What is Cognitive Services?](#)

[Get Started](#)

[Create an account](#)

[Services/APIs](#)

[Academic Knowledge](#)

[Overview](#)

[Knowledge Exploration](#)

[Graph traversal](#)

[API reference](#)

[Bing Autosuggest](#)

[Overview](#)

[Use and Display Requirements](#)

[API reference](#)

[Bing Image Search](#)

[Overview](#)

[Use and Display Requirements](#)

[API reference](#)

[Bing News Search](#)

[Overview](#)

[Use and Display Requirements](#)

[API reference](#)

[Bing Speech](#)

[Overview](#)

[Get started](#)

[Speech recognition API reference](#)

[Text-to-speech API reference](#)

[SDKs](#)

[Bing Spell Check](#)

[Overview](#)

[Use and Display Requirements](#)

[API reference](#)

[Bing Video Search](#)

[Overview](#)

[Use and Display Requirements](#)

[API reference](#)

[Bing Web Search](#)

[Overview](#)

[Use and Display Requirements](#)

[API reference](#)

[Computer Vision](#)

[Overview](#)

[How-to](#)

[Quickstarts](#)

[Tutorials](#)

[API reference](#)

[SDKs](#)

[Samples](#)

[Category taxonomy](#)

[FAQ](#)

[Research papers](#)

[Content Moderator](#)

[Overview](#)

[Get started](#)

[Review tool user guide](#)

[API Reference](#)

[SDKs](#)

[Samples](#)

[FAQ](#)

[Custom Speech Service](#)

[Overview](#)

[Get started](#)

[FAQ](#)

[Glossary](#)

[Emotion](#)

[Overview](#)

[How to](#)

[Quickstarts](#)

[Tutorials](#)

[API Reference for Images](#)

[API Reference for Video](#)

[SDKs](#)

[Samples](#)

[FAQ](#)

[Entity Linking](#)

[Overview](#)

[Get started](#)

[API reference](#)

[SDKs](#)

[Face](#)

[Overview](#)

[How to](#)

[Quickstarts](#)

[Tutorials](#)

[API reference](#)

[SDKs](#)

[Samples](#)

[Glossary](#)

[FAQ](#)

[Release notes](#)

[Knowledge Exploration Service](#)

[Overview](#)

[Getting started](#)

- Command line
- Schema format
- Data format
- Grammar format
- Semantic interpretation
- Web API interface
- Structured query expressions

## Linguistic Analysis

- Overview
- Analyzer names
- Constituency parsing
- Part-of-speech tagging
- Sentences and tokens
- Analyze method
- Analyzers method
- API reference
- SDKs

## Language Understanding Intelligent Services

- Overview
- Plan your app
- Create an app
  - Dashboard
  - Add intents
  - Add utterances
  - Add entities
  - Add features
  - Train + test the app
  - Active learning
  - Publish your app
  - Pre-built entities
  - Cortana prebuilt app
  - Key management

[Create subscription keys](#)

[API reference](#)

[SDKs](#)

[QnAMaker](#)

[Overview](#)

[Quickstart](#)

[Create a knowledge base](#)

[Publish a knowledge base](#)

[Train a knowledge base](#)

[Update a knowledge base](#)

[Authentication](#)

[API Reference](#)

[FAQ](#)

[Recommendations](#)

[Overview](#)

[Recommendations quick start](#)

[Build types + model quality](#)

[Collect training data](#)

[Use the Recommendations UI](#)

[API Reference](#)

[Batch scoring](#)

[Speaker Recognition](#)

[Overview](#)

[API reference](#)

[SDKs](#)

[Text Analytics](#)

[Overview](#)

[Quickstart](#)

[API migration](#)

[API reference](#)

[SDKs](#)

[Translator](#)

[Overview](#)

[Languages](#)

[Hub](#)

[CTF](#)

[Translator Speech API overview](#)

[Translator Speech API reference](#)

[Translator Text API overview](#)

[Translator Text API reference](#)

[Video](#)

[Overview](#)

[Get started in C#](#)

[Call video APIs](#)

[Analyze videos in real time](#)

[API Reference](#)

[SDKs](#)

[Glossary](#)

[Web Language Model](#)

[Overview](#)

[API Reference](#)

[SDKs](#)

[Resources](#)

[Pricing](#)

[Videos](#)

[Service updates](#)

[Code of conduct](#)

# What is Cognitive Services?

4/19/2017 • 2 min to read • [Edit Online](#)

Microsoft Cognitive Services (formerly Project Oxford) are a set of APIs, SDKs and services available to developers to make their applications more intelligent, engaging and discoverable. Microsoft Cognitive Services expands on Microsoft's evolving portfolio of machine learning APIs and enables developers to easily add intelligent features – such as emotion and video detection; facial, speech and vision recognition; and speech and language understanding – into their applications. Our vision is for more personal computing experiences and enhanced productivity aided by systems that increasingly can see, hear, speak, understand and even begin to reason.

You will find documentation of each product and their corresponding API references on the left navigation table.

## Getting started with free trials

Signing up for free trials only takes an email and a few [simple steps](#). You will need a Microsoft Account if you don't already have one. You will receive a unique pair of keys for each API requested. The second one is just a spare. Please do not share the secret keys with anyone. Trials have both rate limit, in terms of transactions per second or minute, and a monthly usage cap. A transaction is simply an API call. You can upgrade to paid tiers to unlock the restrictions.

## Subscription management

Once you are signed in with Microsoft Account you will be able to access [My subscriptions](#) to show the products you are using, the quota remaining and the ability to add additional products to your subscription.

## Upgrade to unlock limits

All the APIs will have a free trial plan. As paid offerings become available for each API you will be directed to the Azure portal to complete the purchase. You can find *Buy* links in your Subscriptions page if you are already using them or you can skip trial altogether and purchase with provided links on [Pricing](#). You will need to set up a Azure subscriber account with a credit card and a phone number. For the first time subscriber you get a \$200 discount (subject to change without notice). If you have a special requirement or simply want to talk to sales, please click "Contact us" button at the top the pricing page.

## Regional availability

The APIs in Cognitive Services are hosted on a growing network of Microsoft-managed data centers. You can find the regional availability for each API in [Azure region list](#).

Looking for a region we don't support yet? Let us know by filing a feature request on our [UserVoice forum](#).

## Support

We're here for you throughout your development process. Reach out to us at any time using the links below.

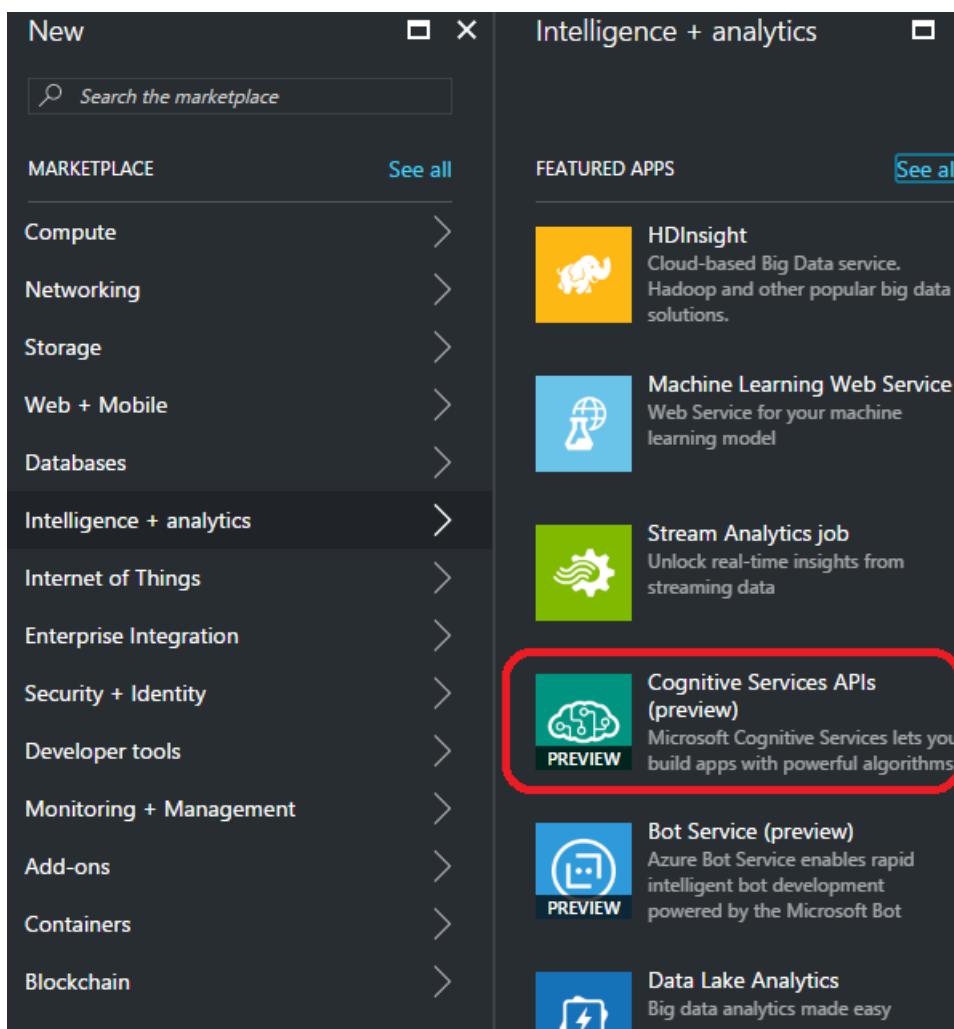
- For support and technical questions, post on [Stack Overflow](#)
- For feedback and feature requests, go to [UserVoice](#)
- For everything else, click "Contact us" on the bottom of any page

# Create a Cognitive Services APIs account in the Azure Portal

3/3/2017 • 2 min to read • [Edit Online](#)

To use Microsoft Cognitive Service APIs, you first need to create an account in the Azure portal.

1. Sign in to the [Azure portal](#).
2. Click **+ NEW**.
3. Select **Intelligence + Analytics** and then **Cognitive Services APIs (preview)**.



4. On the **Create** page, provide the following information:

- **Account Name:** Name of the account. We recommend using a descriptive name, for example `AFaceAPIAccount`.
- **Subscription:** Select one of the available Azure subscriptions in which you have at least Contributor permissions.
- **API Type:** Choose the Cognitive Services API you want to use. For more information about the various Cognitive Services APIs available, please refer to the [Cognitive Services](#) site.

## Academic Knowledge API (preview)

Bing Autosuggest API  
Bing Search APIs  
Bing Speech API  
Bing Spell Check API  
Computer Vision API (preview)  
Content Moderator (preview)  
Custom speech service (Preview)  
Emotion API (preview)  
Face API (preview)  
Language Understanding Intelligent Service (LUIS) (preview)  
Recommendations API (preview)  
Speaker Recognition API (preview)  
Text Analytics API (preview)  
Translator Speech API  
Translator Text API  
Web Language Model API (preview)

- **Pricing tier:** The cost of your Cognitive Services account depends on the actual usage and the options you choose. For more information about pricing for each API, please refer to the [pricing pages](#).
- **Resource Group:** A resource group is a collection of resources that share the same lifecycle, permissions, and policies. To learn more about Resource Groups, see [Manage Azure resources through portal](#).
- **Resource Group Location:** This is required only if the API selected is global (not bound to a location). If the API is global and not bound to a location, however, you must specify a location for the resource group where the metadata associated with the Cognitive Services API account will reside. This location will have no impact on the runtime availability of your account. To learn more about resource group, please refer to [Manage Azure resources through portal](#).
- **API Setting:** By default, account creation is disabled until your [Azure Account Administrator](#) explicitly enables it.

This setting change will apply only to the currently selected API type and location or Resource group location on the panel to the left.

**Create**  
Cognitive Services account - PREVIEW

**Enable or disable account creation for this Azure subscription.**

\* Account name

\* Subscription

\* API type

\* Location

\* Pricing tier ([View full pricing details](#))

\* Resource group ⓘ  
 Create new    Use existing

API Setting ⓘ ! >  
Account creation disabled

Microsoft may use data you send to the Cognitive Services to improve Microsoft products and services. For example we may use content that you provide to the Cognitive Services to improve our underlying algorithms and models over time. Where you send personal data to the Cognitive Services, you are responsible for obtaining sufficient consent from the data subjects. The General Privacy and Security Terms in the [Online Services Terms](#) do not apply to the Cognitive Services.

Please refer to the Microsoft Cognitive Services section in the [Online Services Terms](#) for details.

**Enable** **Disable**

Note: Only the Azure Account Administrator can change this setting. Please refer to the [Create a Cognitive Services account in the Azure portal](#) for details. This setting change will apply only to the currently selected API type and Location on the panel to the left.

#### NOTE

If you receive a notification that the update setting failed, you are not logged in as an [Account Administrator](#). The Account Administrator must follow the previous steps to enable creation.



Update setting failed.

8:19 PM

You are unauthorized to enable Academic Knowledge API (preview) for this subscription, please contact Azure Account Administrator or your company or department IT to help you!

In some cases, the Account Administrator may not have access to the subscription. If so, have the Service Administrator follow the steps in the [Add or change Azure administrator roles that manage the subscription or service](#) document.

To find the Account Administrator or Service Administrator for your subscription, select your subscription in the [Azure portal](#), and then select **Properties**. The **Account Admin** and **Service Admin** information is displayed at the bottom of the properties blade.

The screenshot shows two overlapping windows from the Azure portal. The left window is titled 'Subscriptions' and lists two subscriptions: 'MySubscription...' and 'AnotherSubsc...'. The right window is titled 'MySubscription - Properties' and displays various subscription details. A red box highlights the 'Properties' link under the 'SETTINGS' section. Another red box highlights the 'ACCOUNT ADMIN' and 'SERVICE ADMIN' email fields, both of which are set to 'me@me.com'.

SUBSCRIPTIONS	
<a href="#">+ Add</a>	
Role <span>?</span>	Status <span>?</span>
All	All
<a href="#">Apply</a>	
<input type="text"/> Search to filter items...	
SUBSCRIPT...	SUBSCRIPTION ID
MySubscription...	...
AnotherSubsc...	...

MySubscription - Properties	
<input type="text"/> Search (Ctrl+/)	
<a href="#">Overview</a>	
<a href="#">Access control (IAM)</a>	
<a href="#">Diagnose and solve problems</a>	
<b>BILLING</b>	
<a href="#">Partner information</a>	
<b>SETTINGS</b>	
<a href="#">Programmatic deployment</a>	
<a href="#">Resource groups</a>	
<a href="#">Resources</a>	
<a href="#">Usage + quotas</a>	
<a href="#">Policies</a>	
<a href="#">Management certificates</a>	
<a href="#">My permissions</a>	
<a href="#">Resource providers</a>	
<a href="#">Properties</a>	<b>Properties</b>
<a href="#">Resource locks</a>	
<b>SUPPORT + TROUBLESHOOTING</b>	
<a href="#">New support request</a>	

Service Admin	
<b>SUBSCRIPTION ID</b>	<input type="text"/>
<b>STATUS</b>	<input type="text"/> Active
<b>CURRENT BILLING PERIOD</b>	<input type="text"/> 3/1/2017-3/31/2017
<b>PURCHASE DATE</b>	<input type="text"/> 7/29/2013
<b>CURRENCY</b>	<input type="text"/> USD
<b>OFFER</b>	<input type="text"/>
<b>OFFER ID</b>	<input type="text"/>
<b>ACCOUNT ADMIN</b>	<input type="text"/> me@me.com
<b>SERVICE ADMIN</b>	<input type="text"/> me@me.com

Microsoft may use data you send to the Cognitive Services to improve Microsoft products and services. For more information, please refer to the [Microsoft Cognitive Services section](#) in the Online Services Terms.

5. To pin the account to the Azure portal dashboard, click **Pin to Dashboard**.
6. Click **Create** to create the account.

After the Cognitive Services account is successfully deployed, click the tile in the dashboard to view the account information.

You can use the **Endpoint URL** in the **Overview** section and keys in the **Keys** section to start making API calls in your applications.

The screenshot shows the 'bingSearch' Cognitive Services account in the Azure portal. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Keys, Pricing tier, Properties, Locks, and Automation script. The main area displays the 'Overview' section with details like Resource group (bvttest), Status (Active), Location (global), Subscription name (Dev-01), Subscription ID, and API type (Bing Search APIs). The Endpoint is listed as https://api.cognitive.microsoft.com/bing/v5.0. A red box highlights the Endpoint URL.

The screenshot shows the 'bingSearch - Keys' page. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Keys, Pricing tier, Properties, Locks, and Automation script. The main area shows two key slots: 'KEY 1' and 'KEY 2', each with a regenerate button. The account name is set to 'bingSearch'. A notice at the top states: 'Notice: It may take up to 10 minutes for the newly (re)generated keys to take effect.' A red box highlights the 'Keys' link in the sidebar.

## Next Steps

- For more information about all the Microsoft Cognitive Services available, see [Cognitive Services](#).
- For quick start guides to using some example Cognitive Services APIs, see:
  - [Getting started with the Text Analytics APIs to detect sentiment, key phrases, topics and language](#)
  - [Quick start guide for the Cognitive Services Recommendations API](#)

# Academic Knowledge API

4/12/2017 • 2 min to read • [Edit Online](#)

Welcome to the Academic Knowledge API. With this service, you will be able to interpret user queries for academic intent and retrieve rich information from the Microsoft Academic Graph (MAG). The MAG knowledge base is a web-scale heterogeneous entity graph comprised of entities that model scholarly activities: field of study, author, institution, paper, venue, and event.

The MAG data is mined from the Bing web index as well as an in-house knowledge base from Bing. As a result of on-going Bing indexing, this API will contain fresh information from the Web following discovery and indexing by Bing. Based on this dataset, the Academic Knowledge APIs enables a knowledge-driven, interactive dialog that seamlessly combines reactive search with proactive suggestion experiences, rich research paper graph search results, and histogram distributions of the attribute values for a set of papers and related entities.

For more information on the Microsoft Academic Graph, see <http://aka.ms/academicgraph>.

## Features

The Academic Knowledge API consists of four related REST endpoints:

1. **interpret** – Interprets a natural language user query string. Returns annotated interpretations to enable rich search-box auto-completion experiences that anticipate what the user is typing.
2. **evaluate** – Evaluates a query expression and returns Academic Knowledge entity results.
3. **calchistogram** – Calculates a histogram of the distribution of attribute values for the academic entities returned by a query expression, such as the distribution of citations by year for a given author.
4. **graph search** – Searches for a given graph pattern and returns the matched entity results.

Used together, these API methods allow you to create a rich semantic search experience. Given a user query string, the **interpret** method provides you with an annotated version of the query and a structured query expression, while optionally completing the user's query based on the semantics of the underlying academic data. For example, if a user types the string *latent s*, the **interpret** method can provide a set of ranked interpretations, suggesting that the user might be searching for the field of study *latent semantic analysis*, the paper *latent structure analysis*, or other entity expressions starting with *latent s*. This information can be used to quickly guide the user to the desired search results.

The **evaluate** method can be used to retrieve a set of matching paper entities from the academic knowledge base, and the **calchistogram** method can be used to calculate the distribution of attribute values for a set of paper entities which can be used to further filter the search results.

The **graph search** method has two modes: *json* and *lambda*. The *json* mode can perform graph pattern matching according to the graph patterns specified by a JSON object. The *lambda* mode can perform server-side computations during graph traversals according to the user-specified lambda expressions.

## Getting Started

Please see the subtopics at the left for detailed documentation. Note that to improve the readability of the examples, the REST API calls contain characters (such as spaces) that have not been URL-encoded. Your code will need to apply the appropriate URL-encodings.

# Entity Attributes

4/12/2017 • 1 min to read • [Edit Online](#)

The academic graph is composed of 7 types of entity. All entities will have a Entity ID and a Entity type.

## Common Entity Attributes

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
Ty	Entity type	enum	Equals

## Entity type enum

NAME	VALUE
Paper	0
Author	1
Journal	2
Conference Series	3
Conference Instance	4
Affiliation	5
Field Of Study	6

# Paper Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to paper entity. (Ty = '1')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
Ti	Paper title	String	Equals, StartsWith
L	Paper language code seperated by "@@@"	String	Equals
Y	Paper year	Int32	Equals, IsBetween
D	Paper date	Date	Equals, IsBetween
CC	Citation count	Int32	none
ECC	Estimated citation Count	Int32	none
AA.AuN	Author name	String	Equals, StartsWith
AA.Auld	Author ID	Int64	Equals
AA.Afn	Author affiliation name	String	Equals, StartsWith
AA.Afld	Author affiliation ID	Int64	Equals
AAS	Author order for the paper	Int32	Equals
F.FN	Field of study name	String	Equals, StartsWith
F.Fld	Field of study ID	Int64	Equals
J.JN	Journal name	String	Equals, StartsWith
J.JId	Journal ID	Int64	Equals
C.CN	Conference series name	String	Equals, StartsWith
C.CId	Conference series ID	Int64	Equals

NAME	DESCRIPTION	TYPE	OPERATIONS
RId	Referenced papers' ID	Int64[]	Equals
W	Words from paper title and Abstract	String[]	Equals
E	Extended metadata (see table below)	String	none

## Extended Metadata Attributes

NAME	DESCRIPTION
DN	Display Name of the paper
S	Sources - list of web sources of the paper, sorted by static rank
S.Ty	Source Type (1:HTML, 2:Text, 3:PDF, 4:DOC, 5:PPT, 6:XLS, 7:PS)
S.U	Source URL
VFN	Venue Full Name - full name of the Journal or Conference
VSN	Venue Short Name - short name of the Journal or Conference
V	Volume - journal volume
I	Issue - journal issue
FP	FirstPage - first page of paper
LP	LastPage - last page of paper
DOI	Digital Object Identifier
CC	Citation Contexts – List of referenced paper ID's and the corresponding context in the paper (e.g. [{123:[“brown foxes are known for jumping as referenced in paper 123”, “the lazy dogs are a historical misnomer as shown in paper 123”]}])
IA	Inverted Abstract
IA.IndexLength	Number of items in the index (abstract's word count)
IA.InvertedIndex	List of abstract words and their corresponding position in the original abstract (e.g. [{"the":0, 15, 30}, {"brown":1}, {"fox":2}]))

# Author Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to author entity. (Ty = '1')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
AuN	Author normalized name	String	Equals
DAuN	Author display name	String	none
CC	Author total citation count	Int32	none
ECC	Author total estimated citation count	Int32	none
E	Extended metadata (see table below)	String	none
SSD	Satori data	String	none

## Extended Metadata Attributes

NAME	DESCRIPTION
LKA.Afn	affiliation's display name associated with the author
LKA.Afld	affiliation's entity ID associated with the author

# Affiliation Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to affiliation entity. (Ty = '5')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
AfN	Affiliation normalized name	String	Equals
DAfN	Affiliation display name	String	none
CC	Affiliation total citation count	Int32	none
ECC	Affiliation total estimated citation count	Int32	none
SSD	Satori data	String	none

# Field Of Study Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to field of study entity. (Ty = '6')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
FN	Field of study normalized name	String	Equals
DFN	Field of study display name	String	none
CC	Field of study total citation count	Int32	none
ECC	Field of total estimated citation count	Int32	none
FL	Level in fields of study hierarchy	Int32	Equals, IsBetween
FP.FN	Parent field of study name	String	Equals
FP.FId	Parent field of study ID	Int64	Equals
SSD	Satori data	String	none

# Conference Series Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to conference series entity. (Ty = '3')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
CN	Conference series normalized name	String	Equals
DCN	Conference series display name	String	none
CC	Conference series total citation count	Int32	none
ECC	Conference series total estimated citation count	Int32	none
F.IId	Field of study entity ID associated with the conference series	Int64	Equals
F.FN	Field of study name associated with the conference series	Equals, StartsWith	
SSD	Satori data	String	none

# Conference Instance Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to conference instance entity. (Ty = '4')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
CIN	Conference instance normalized name ({{ConferenceSeriesNormalize dName} {ConferenceInstanceYear}})	String	Equals
DCN	Conference instance display name ({{ConferenceSeriesName} : {ConferenceInstanceYear}})	String	none
CIL	Location of the conference instance	String	Equals, StartsWith
CISD	Start date of the conference instance	Date	Equals, IsBetween
CIED	End date of the conference instance	Date	Equals, IsBetween
CIARD	Abstract registration due date of the conference instance	Date	Equals, IsBetween
CISDD	Submission due date of the conference instance	Date	Equals, IsBetween
CIFVD	Final version due date of the conference instance	Date	Equals, IsBetween
CINDD	Notification date of the conference instance	Date	Equals, IsBetween
CD.T	Title of a conference instance event	Date	Equals, IsBetween
CD.D	Date of a conference instance event	Date	Equals, IsBetween
PCS.CN	Conference series name of the instance	String	Equals

NAME	DESCRIPTION	TYPE	OPERATIONS
PCS.CId	Conference series ID of the instance	Int64	Equals
CC	Conference instance total citation count	Int32	none
ECC	Conference instance total estimated citation count	Int32	none
SSD	Satori data	String	none

## Extended Metadata Attributes

NAME	DESCRIPTION
FN	Conference instance full name

# Journal Entity

4/12/2017 • 1 min to read • [Edit Online](#)

\*Below attributes are specific to journal entity. (Ty = '2')

NAME	DESCRIPTION	TYPE	OPERATIONS
Id	Entity ID	Int64	Equals
DJN	Journal normalized name	String	none
JN	Journal display name	String	Equals
CC	Journal total citation count	Int32	none
ECC	Journal total estimated citation count	Int32	none
SSD	Satori data	String	none

# CalcHistogram Method

4/12/2017 • 2 min to read • [Edit Online](#)

The **calchistogram** REST API is used to calculate the distribution of attribute values for a set of paper entities.

## REST endpoint:

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/calchistogram?
```

## Request Parameters

NAME	VALUE	REQUIRED?	DESCRIPTION
<b>expr</b>	Text string	Yes	A query expression that specifies the entities over which to calculate histograms.
<b>model</b>	Text string	No	Select the name of the model that you wish to query. Currently, the value defaults to <i>latest</i> .
<b>attributes</b>	Text string	No default:	A comma-delimited list that specifies the attribute values that are included in the response. Attribute names are case-sensitive.
<b>count</b>	Number	No Default: 10	Number of results to return.
<b>offset</b>	Number	No Default: 0	Index of the first result to return.

## Response (JSON)

NAME	DESCRIPTION
<b>expr</b>	The expr parameter from the request.
<b>num_entities</b>	Total number of matching entities.
<b>histograms</b>	An array of histograms, one for each attribute specified in the request.

NAME	DESCRIPTION
<b>histograms[x].attribute</b>	Name of the attribute over which the histogram was computed.
<b>histograms[x].distinct_values</b>	Number of distinct values among matching entities for this attribute.
<b>histograms[x].total_count</b>	Total number of value instances among matching entities for this attribute.
<b>histograms[x].histogram</b>	Histogram data for this attribute.
<b>histograms[x].histogram[y].value</b>	A value for the attribute.
<b>histograms[x].histogram[y].logprob</b>	Total natural log probability of matching entities with this attribute value.
<b>histograms[x].histogram[y].count</b>	Number of matching entities with this attribute value.
<b>aborted</b>	True if the request timed out.

**Example:**

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/calchistogram?expr=And(Composite(AA.AuN=='jaime
teevan'),Y>2012)&attributes=Y,F.FN&count=4
```

In this example, in order to generate a histogram of the count of publications by year for a particular author since 2010, we can first generate the query expression using the **interpret** API with query string: *papers by jaime teevan after 2012*.

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/interpret?query=papers by jaime teevan after 2012
```

The expression in the first interpretation that is returned from the interpret API is *And(Composite(AA.AuN=='jaime teevan'),Y>2012)*.

This expression value is then passed in to the **calchistogram** API. The *attributes=Y,F.FN* parameter indicates that the distributions of paper counts should be by Year and Field of Study, e.g.:

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/calchistogram?expr=And(Composite(AA.AuN=='jaime
teevan'),Y>2012)&attributes=Y,F.FN&count=4
```

The response to this request first indicates that there are 37 papers that match the query expression. For the *Year* attribute, there are 3 distinct values, one for each year after 2012 (i.e. 2013, 2014, and 2015) as specified in the query. The total paper count over the 3 distinct values is 37. For each *Year*, the histogram shows the value, total natural log probability, and count of matching entities.

The histogram for *Field of Study* shows that there are 34 distinct fields of study. As a paper may be associated with multiple fields of study, the total count (53) can be larger than the number of matching entities. Although there are

34 distinct values, the response only includes the top 4 because of the `count=4` parameter.

```
{  
  "expr": "And(Composite(AA.AuN=='jaime teeven'),Y>2012)",  
  "num_entities": 37,  
  "histograms": [  
    {  
      "attribute": "Y",  
      "distinct_values": 3,  
      "total_count": 37,  
      "histogram": [  
        {  
          "value": 2014,  
          "logprob": -15.753,  
          "count": 15  
        },  
        {  
          "value": 2013,  
          "logprob": -15.805,  
          "count": 12  
        },  
        {  
          "value": 2015,  
          "logprob": -16.035,  
          "count": 10  
        }  
      ]  
    },  
    {  
      "attribute": "F.FN",  
      "distinct_values": 34,  
      "total_count": 53,  
      "histogram": [  
        {  
          "value": "crowdsourcing",  
          "logprob": -15.258,  
          "count": 9  
        },  
        {  
          "value": "information retrieval",  
          "logprob": -16.002,  
          "count": 4  
        },  
        {  
          "value": "personalization",  
          "logprob": -16.226,  
          "count": 3  
        },  
        {  
          "value": "mobile search",  
          "logprob": -17.228,  
          "count": 2  
        }  
      ]  
    }  
  ]  
}
```

# Evaluate Method

4/12/2017 • 1 min to read • [Edit Online](#)

The **evaluate** REST API is used to return a set of academic entities based on a query expression.

## REST endpoint:

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/evaluate?
```

## Request Parameters

NAME	VALUE	REQUIRED?	DESCRIPTION
<b>expr</b>	Text string	Yes	A query expression that specifies which entities should be returned.
<b>model</b>	Text string	No	Name of the model that you wish to query. Currently, the value defaults to <i>latest</i> .
<b>attributes</b>	Text string	No default: Id	A comma-delimited list that specifies the attribute values that are included in the response. Attribute names are case-sensitive.
<b>count</b>	Number	No Default: 10	Number of results to return.
<b>offset</b>	Number	No Default: 0	Index of the first result to return.
<b>orderby</b>	Text string	No Default: by decreasing prob	Name of an attribute that is used for sorting the entities. Optionally, ascending/descending can be specified. The format is: <i>name:asc</i> or <i>name:desc</i> .

## Response (JSON)

NAME	DESCRIPTION
<b>expr</b>	The <i>expr</i> parameter from the request.

NAME	DESCRIPTION
<b>entities</b>	An array of 0 or more entities that matched the query expression. Each entity contains a natural log probability value and the values of other requested attributes.
<b>aborted</b>	True if the request timed out.

**Example:**

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/evaluate?expr=
Composite(AA.AuN=='jaime teeven')&count=2&attributes=Ti,Y,CC,AA.AuN,AA.AuId
```

Typically, an expression will be obtained from a response to the **interpret** method. But you can also compose query expressions yourself (see [Query Expression Syntax](#)).

Using the *count* and *offset* parameters, a large number of results may be obtained without sending a single request that results in a huge (and potentially slow) response. In this example, the request used the expression for the first interpretation from the **interpret** API response as the *expr* value. The *count=2* parameter specifies that 2 entity results are being requested. And the *attributes=Ti,Y,CC,AA.AuN,AA.AuId* parameter indicates that the title, year, citation count, author name, and author ID are requested for each result. See [Entity Attributes](#) for a list of attributes.

```
{
  "expr": "Composite(AA.AuN=='jaime teeven')",
  "entities": [
    [
      {
        "logprob": -15.08,
        "Ti": "personalizing search via automated analysis of interests and activities",
        "Y": 2005,
        "CC": 372,
        "AA": [
          {
            "AuN": "jaime teeven",
            "AuId": 1968481722
          },
          {
            "AuN": "susan t dumais",
            "AuId": 676500258
          },
          {
            "AuN": "eric horvitz",
            "AuId": 1470530979
          }
        ]
      },
      {
        "logprob": -15.389,
        "Ti": "the perfect search engine is not enough a study of orienteering behavior in directed search",
        "Y": 2004,
        "CC": 237,
        "AA": [
          {
            "AuN": "jaime teeven",
            "AuId": 1982462162
          },
          {
            "AuN": "christine alvarado",
            "AuId": 2163512453
          },
          {
            "AuN": "mark s ackerman",
            "AuId": 2055132526
          },
          {
            "AuN": "david r karger",
            "AuId": 2012534293
          }
        ]
      }
    ]
  }
}
```

# Interpret Method

4/12/2017 • 2 min to read • [Edit Online](#)

The **interpret** REST API takes an end user query string (i.e., a query entered by a user of your application) and returns formatted interpretations of user intent based on the Academic Graph data and the Academic Grammar.

To provide an interactive experience, you can call this method repeatedly after each character entered by the user. In that case, you should set the **complete** parameter to 1 to enable auto-complete suggestions. If your application does not need auto-completion, you should set the **complete** parameter to 0.

## REST endpoint:

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/interpret?
```

## Request Parameters

NAME	VALUE	REQUIRED?	DESCRIPTION
<b>query</b>	Text string	Yes	Query entered by user. If complete is set to 1, query will be interpreted as a prefix for generating query auto-completion suggestions.
<b>model</b>	Text string	No	Name of the model that you wish to query. Currently, the value defaults to <i>latest</i> .
<b>complete</b>	0 or 1	No default:0	1 means that auto-completion suggestions are generated based on the grammar and graph data.
<b>count</b>	Number	No default:10	Maximum number of interpretations to return.
<b>offset</b>	Number	No default:0	Index of the first interpretation to return. For example, <i>count=2&amp;offset=0</i> returns interpretations 0 and 1. <i>count=2&amp;offset=2</i> returns interpretations 2 and 3.
<b>timeout</b>	Number	No default:1000	Timeout in milliseconds. Only interpretations found before the timeout has elapsed are returned.

## Response (JSON)

NAME	DESCRIPTION
<b>query</b>	The <i>query</i> parameter from the request.
<b>interpretations</b>	An array of 0 or more different ways of matching user input against the grammar.
<b>interpretations[x].logprob</b>	The relative natural log probability of the interpretation. Larger values are more likely.
<b>interpretations[x].parse</b>	An XML string that shows how each part of the query was interpreted.
<b>interpretations[x].rules</b>	An array of 1 or more rules defined in the grammar that were invoked during interpretation. For the Academic Knowledge API, there will always be 1 rule.
<b>interpretations[x].rules[y].name</b>	Name of the rule.
<b>interpretations[x].rules[y].output</b>	Output of the rule.
<b>interpretations[x].rules[y].output.type</b>	The data type of the output of the rule. For the Academic Knowledge API, this will always be "query".
<b>interpretations[x].rules[y].output.value</b>	The output of the rule. For the Academic Knowledge API, this is a query expression string that can be passed to the evaluate and calchistogram methods.
<b>aborted</b>	True if the request timed out.

**Example:**

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/interpret?query=papers by jaime&complete=1&count=2
```

The response below contains the top two (because of the parameter *count*=2) most likely interpretations that complete the partial user input *papers by jaime*: *papers by jaime teevan* and *papers by jaime green*. The service generated query completions instead of considering only exact matches for the author *jaime* because the request specified *complete*=1. Note that the canonical value *jl green* matched via the synonym *jamie green*, as indicated in the parse.

```
{
  "query": "papers by jaime",
  "interpretations": [
    {
      "logprob": -12.728,
      "parse": "<rule name=\"#GetPapers\">papers by <attr name=\"academic#AA.AuN\">jaime teevan</attr></rule>",
      "rules": [
        {
          "name": "#GetPapers",
          "output": {
            "type": "query",
            "value": "Composite(AA.AuN=='jaime teevan')"
          }
        }
      ]
    },
    {
      "logprob": -12.774,
      "parse": "<rule name=\"#GetPapers\">papers by <attr name=\"academic#AA.AuN\" canonical=\"j 1 green\">jaime green</attr></rule>",
      "rules": [
        {
          "name": "#GetPapers",
          "output": {
            "type": "query",
            "value": "Composite(AA.AuN=='j 1 green')"
          }
        }
      ]
    }
  ]
}
```

To retrieve entity results for an interpretation, use `output.value` from the **interpret** API, and pass that into the **evaluate** API via the `expr` parameter. In this example, the query for the first interpretation is:

```
evaluate?expr=Composite(AA.AuN=='jaime teevan')
```

# Similarity Method

4/12/2017 • 1 min to read • [Edit Online](#)

The **similarity** REST API is used to calculate the academic similarity between two strings.

## REST endpoint:

```
https://westus.api.cognitive.microsoft.com/academic/v1.0/similarity?
```

## Request Parameters

PARAMETER	DATA TYPE	REQUIRED	DESCRIPTION
s1	String	Yes	String* to be compared
s2	String	Yes	String* to be compared

\*Strings to compare have a maximum length of 1MB.

## Response

NAME	DESCRIPTION
<b>SimilarityScore</b>	A floating point value representing the cosine similarity of s1 and s2, with values closer to 1.0 meaning more similar and values closer to -1.0 meaning less

## Success/Error Conditions

HTTP STATUS	REASON	RESPONSE
<b>200</b>	Success	Floating point number
<b>400</b>	Bad request or request invalid	Error message
<b>500</b>	Internal server error	Error message
<b>Timed out</b>	Request timed out.	Error message

## Example: Calculate similarity of two partial abstracts

### Request:

[https://westus.api.cognitive.microsoft.com/academic/v1.0/similarity?&s1=Using complementary priors, we derive a fast greedy algorithm that can learn deep directed belief networks one layer at a time, provided the top two layers form an undirected associative memory&&s2=Deepneural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks](https://westus.api.cognitive.microsoft.com/academic/v1.0/similarity?&s1=Using%20complementary%20priors,%20we%20derive%20a%20fast%20greedy%20algorithm%20that%20can%20learn%20deep%20directed%20belief%20networks%20one%20layer%20at%20a%20time,%20provided%20the%20top%20two%20layers%20form%20an%20undirected%20associative%20memory&&s2=Deepneural%20nets%20with%20a%20large%20number%20of%20parameters%20are%20very%20powerful%20machine%20learning%20systems.%20However,%20overfitting%20is%20a%20serious%20problem%20in%20such%20networks)

In this example, we generate the similarity score between two partial abstracts using the **similarity** API.

**Response:**

0.520

**Remarks:**

The similarity score is determined by assessing the academic concepts through word embedding. In this example, 0.52 means that the two partial abstracts are somewhat similar.

# Query Expression Syntax

4/12/2017 • 3 min to read • [Edit Online](#)

We have seen that the response to an **interpret** request includes a query expression. The grammar that interpreted the user's query created a query expression for each interpretation. A query expression can then be used to issue an **evaluate** request to retrieve entity search results.

You can also construct your own query expressions and use them in an **evaluate** request. This can be useful if you are building your own user interface which creates a query expression in response to the user's actions. To do this, you need to know the syntax for query expressions.

Each entity attribute that can be included in a query expression has a specific data type and a set of possible query operators. The set of entity attributes and supported operators for each attribute is specified in [Entity Attributes](#). A single-value query requires the attribute to support the *Equals* operation. A prefix query requires the attribute to support the *StartsWith* operation. Numeric range queries require the attribute to support the *IsBetween* operation.

Some of the entity data are stored as composite attributes, as indicated by a dot '.' in the attribute name. For example, Author/Affiliation information is represented as a composite attribute. It contains 4 components: AuN, Auld, AfN, Afld. These components are separate pieces of data that form a single entity attribute value.

## **String Attribute: Single value** (includes matches against synonyms)

Ti='indexing by latent semantic analysis'

Composite(AA.AuN='sue dumais')

## **String Attribute: Exact single value** (matches only canonical values)

Ti=='indexing by latent semantic analysis'

Composite(AA.AuN=='susan t dumais')

## **String Attribute: Prefix value**

Ti='indexing by latent seman'...

Composite(AA.AuN='sue du'...)

## **Numeric Attribute: Single value**

Y=2010

## **Numeric Attribute: Range value**

Y>2005

Y>=2005

Y<2010

Y<=2010

Y=[2010, 2012) (includes only left boundary value: 2010, 2011)

Y=[2010, 2012] (includes both boundary values: 2010, 2011, 2012)

## **Numeric Attribute: Prefix value**

Y='19'... (any numeric value that starts with 19)

## **Date Attribute: Single value**

D='2010-02-04'

## **Date Attribute: Range value**

D>'2010-02-03'

D=['2010-02-03','2010-02-05']

## **And/Or Queries:**

```
And(Y=1985, Ti='disordered electronic systems')
Or(Ti='disordered electronic systems', Ti='fault tolerance principles and practice')
And(Or(Y=1985,Y=2008), Ti='disordered electronic systems')
```

### Composite Queries:

To query components of a composite attribute, you need to enclose the part of the query expression that refers to the composite attribute in the Composite() function.

For example, to query for papers by author name, use the following query:

```
Composite(AA.AuN='mike smith')
```

To query for papers by a particular author while the author was at a particular institution, use the following query:

```
Composite(And(AA.AuN='mike smith',AA.AfN='harvard university'))
```

The Composite() function ties the two parts of the composite attribute together. This means that we only get papers where one of the authors is "Mike Smith" while he was at Harvard.

To query for papers by a particular author in affiliations with (other) authors from a particular institution, use the following query:

```
And(Composite(AA.AuN='mike smith'),Composite(AA.AfN='harvard university'))
```

In this version, because Composite() is applied to the author and affiliation individually before And(), we get all papers where one of the authors is "Mike Smith" and one of the authors' affiliations is "Harvard". This sounds similar to the previous query example, but it's not the same thing.

In general, consider the following example: We have a composite attribute C that has two components A and B. An entity may have multiple values for C. These are our entities:

```
E1: C={A=1, B=1}  C={A=1,B=2}  C={A=2,B=3}
E2: C={A=1, B=3}  C={A=3,B=2}
```

The query

```
Composite(And(C.A=1, C.B=2))
```

matches only entities that have a value for C where the component C.A is 1 and the component C.B is 2. Only E1 matches this query.

The query

```
And(Composite(C.A=1), Composite(C.B=2))
```

matches entities that have a value for C where C.A is 1 and also have a value for C where C.B is 2. Both E1 and E2

match this query.

Please note:

- You cannot reference a part of a composite attribute outside of a Composite() function.
- You cannot reference parts of two different composite attributes inside the same Composite() function.
- You cannot reference an attribute that is not part of a composite attribute inside a Composite() function.

# Lambda Search Syntax

4/12/2017 • 3 min to read • [Edit Online](#)

Each *lambda* search query string describes a graph pattern. A query must have at least one starting node, specifying from which graph node we start the traversal. To specify a starting node, call the *MAG.StartFrom()* method and pass in the ID(s) of one or more node(s) or a query object that specifies the search constraints. The *StartFrom()* method has three overloads. All of them take two arguments with the second being optional. The first argument can be a long integer, an enumerable collection of long integer, or a string representing a JSON object, with the same semantics as in *json* search:

```
StartFrom(long cellid, IEnumerable<string> select = null)
StartFrom(IEnumerable<long> cellid, IEnumerable<string> select = null)
StartFrom(string queryObject, IEnumerable<string> select = null)
```

The process of writing a lambda search query is to walk from one node to another. To specify the type of the edge to walk through, use *FollowEdge()* and pass in the desired edge types. *FollowEdge()* takes an arbitrary number of string arguments:

```
FollowEdge(params string[] edgeTypes)
```

## NOTE

If we do not care about the type(s) of the edge(s) to follow, simply omit *FollowEdge()* between two nodes: the query will walk through all the possible edges between these two nodes.

We can specify the traversal actions to be taken on a node via *VisitNode()*, that is, whether to stop at this node and return the current path as the result or to continue to explore the graph. The enum type *Action* defines two types of actions: *Action.Return* and *Action.Continue*. We can pass such an enum value directly into *VisitNode()*, or combine them with bitwise-and operator '&'. When two action are combined, it means that both actions will be taken. Note: do not use bitwise-or operator '|' on actions. Doing so will cause the query to terminate without returning anything. Skipping *VisitNode()* between two *FollowEdge()* calls will cause the query to unconditionally explore the graph after arriving at a node.

```
VisitNode(Action action, IEnumerable<string> select = null)
```

For *VisitNode()*, we can also pass in a lambda expression of type *Expression<Func<INode, Action>>*, which takes an *INode* and returns a traversal action:

```
VisitNode(Expression<Func<INode, Action>> action, IEnumerable<string> select = null)
```

# INode

*INode* provides *read-only* data access interfaces and a few built-in helper functions on a node.

## Basic data access interfaces

`long CellID`

The 64-bit ID of the node.

```
T GetField<T>(string fieldName)
```

Gets the value of the specified property. *T* is the desired type that the field is supposed to be interpreted as. Automatic type casting will be attempted if the desired type cannot be implicitly converted from the type of the field. Note: when the property is multi-valued, *GetField<string>* will cause the list to be serialized to a Json string ["val1", "val2", ...]. If the property does not exist, it will throw an exception and abort the current graph exploration.

```
bool ContainsField(string fieldName)
```

Tells if a field with the given name exists in the current node.

```
string get(string fieldName)
```

Works like *GetField<string>(fieldName)*. However, it does not throw exceptions when the field is not found, it returns an empty string("") instead.

```
bool has(string fieldName)
```

Tells if the given property exists in the current node. Same as *ContainsField(fieldName)*.

```
bool has(string fieldName, string value)
```

Tells if the property has the given value.

```
int count(string fieldName)
```

Get the count of values of a given property. When the property does not exist, returns 0.

## Built-in Helper Functions

```
Action return_if(bool condition)
```

Returns *Action.Return* if the condition is *true*. If the condition is *false* and this expression is not joined with other actions by a bitwise-and operator, the graph exploration will be aborted.

```
Action continue_if(bool condition)
```

Returns *Action.Continue* if the condition is *true*. If the condition is *false* and this expression is not joined with other actions by a bitwise-and operator, the graph exploration will be aborted.

```
bool dice(double p)
```

Generates a random number that is greater than or equal to 0.0 and less than 1.0. This function returns *true* only if the number is less than or equal to *p*.

Compared with *json* search, *lambda* search is more expressive: C# lambda expressions can be directly used to specify query patterns. Here are two examples.

```
MAG.StartFrom(@"
    type : ""ConferenceSeries"",
    match : {
        FullName : ""graph""
    }
}", new List<string>{ "FullName", "ShortName" })
.FollowEdge("ConferenceInstanceIDs")
.VisitNode(v => v.return_if(v.GetField<DateTime>("StartDate").ToString().Contains("2014")),
    new List<string>{ "FullName", "StartDate" })
```

```
MAG.StartFrom(@"
    type : ""Affiliation"",
    match : {
        Name : ""microsoft""
    }
").FollowEdge("PaperIDs")
.VisitNode(v => v.return_if(v.get("NormalizedTitle").Contains("graph") || v.GetField<int>("CitationCount") >
100),
    new List<string>{ "OriginalTitle", "CitationCount" })
```

# Graph Search Method

4/12/2017 • 2 min to read • [Edit Online](#)

The **graph search** REST API is used to return a set of academic entities based on the given graph patterns. The response is a set of graph paths satisfying the user-specified constraints. A graph path is an interleaved sequence of graph nodes and edges in the form of  $v_0, e_0, v_1, e_1, \dots, v_n$ , where  $v_0$  is the starting node of the path.

## REST endpoint:

```
https://westus.api.cognitive.microsoft.com/academic/graph/v1.0/search?
```

## Request Parameters

NAME	VALUE	REQUIRED?	DESCRIPTION
<b>mode</b>	Text string	Yes	Name of the mode that you wish to use. The value is either <i>json</i> or <i>lambda</i> .

The graph search method must be called via an HTTP POST request. The post request should include the content type header: **application/json**.

### JSON Search

For the *json* search, the POST body is a JSON object. The JSON object describes a path pattern with user-specified constraints (see the [specification of JSON object](#) for *json* search).

### Lambda Search

For the *lambda* search, the POST body is a plain-text string. The POST body is a LIKQ lambda query string, which is a single C# statement (see the [specification of query string](#) for *lambda* search).

## Response (JSON)

NAME	DESCRIPTION
<b>results</b>	An array of 0 or more entities that match the query expression. Each entity contains the values of requested attributes. This field is present if the request has been successfully processed.
<b>error</b>	HTTP status codes. This field is present if the request fails.
<b>message</b>	Error message. This field is present if the request fails.

If a query cannot be processed within *800 ms*, a *timeout* error will be returned.

### Example:

#### JSON Search

```
https://westus.api.cognitive.microsoft.com/academic/graph/v1.0/search?mode=json
```

For the *json* search, if we want to get the papers whose titles contain "graph engine" and written by "bin shao", we can specify the query as follows.

```
{  
    "path": "/paper/AuthorIDs/author",  
    "paper": {  
        "type": "Paper",  
        "NormalizedTitle": "graph engine",  
        "select": [  
            "OriginalTitle"  
        ]  
    },  
    "author": {  
        "return": {  
            "type": "Author",  
            "Name": "bin shao"  
        }  
    }  
}
```

The output of a query is an array of graph paths. A graph path is an array of node objects corresponding to the nodes specified in the query path. These node objects have at least one property *CellID*, which represents the entity ID. Other properties can be retrieved by specifying the property names via the select operator of a *Traversal Action Object*.

```
{
  "Results": [
    [
      {
        "CellID": 2160459668,
        "OriginalTitle": "Trinity: a distributed graph engine on a memory cloud"
      },
      {
        "CellID": 2093502026
      }
    ],
    [
      {
        "CellID": 2171539317,
        "OriginalTitle": "A distributed graph engine for web scale RDF data"
      },
      {
        "CellID": 2093502026
      }
    ],
    [
      {
        "CellID": 2411554868,
        "OriginalTitle": "A distributed graph engine for web scale RDF data"
      },
      {
        "CellID": 2093502026
      }
    ],
    [
      {
        "CellID": 73304046,
        "OriginalTitle": "The Trinity graph engine"
      },
      {
        "CellID": 2093502026
      }
    ]
  ]
}
```

#### Lambda Search

<https://westus.api.cognitive.microsoft.com/academic/graph/v1.0/search?mode=lambda>

For the *lambda* search, if we want to get the author IDs of a given paper, we can write a query like the following one.

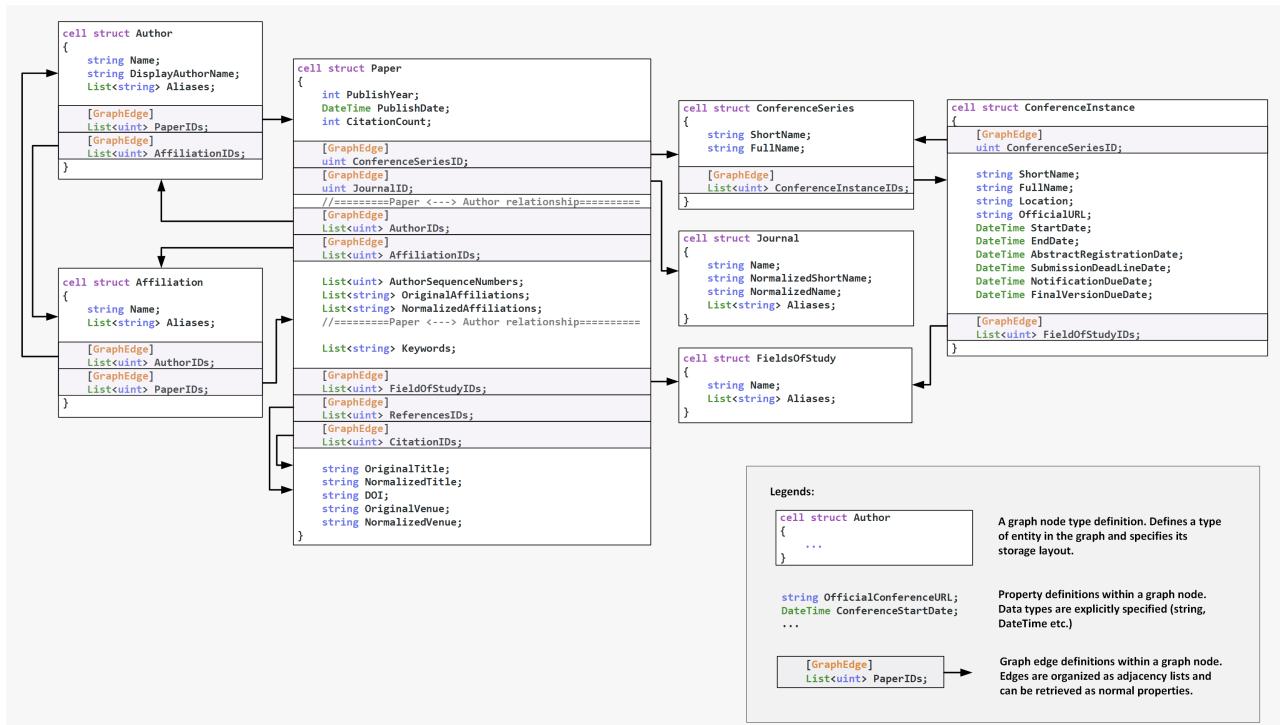
```
MAG.StartFrom(@"{
  type : ""Paper"",
  match : {
    NormalizedTitle : ""trinity: a distributed graph engine on a memory cloud""
  }
}).FollowEdge("AuthorIDs").VisitNode(Action.Return)
```

The output of a *lambda* search query is also an array of graph paths:

```
{
  "Results": [
    [
      {
        "CellID": 2160459668
      },
      {
        "CellID": 2142490828
      }
    ],
    [
      {
        "CellID": 2160459668
      },
      {
        "CellID": 2116756368
      }
    ],
    [
      {
        "CellID": 2160459668
      },
      {
        "CellID": 2093502026
      }
    ]
  ]
}
```

## Graph Schema

Graph schema is useful for writing graph search queries. It is shown in the following figure.



# JSON Search Syntax

4/12/2017 • 3 min to read • [Edit Online](#)

```
/* Query Object:  
Suppose we have a query path /v0/e0/v1/e1/...  
A query object contains constraints to be applied to graph nodes in a path.  
Constraints are given in <"node identifier", {constraint object}> key-value pairs:  
*/  
{  
    /* query path can also be set in the query object */  
    path: "/v0/e0/v1/e1/.../vn/",  
    v0: { /* A Constraint Object */ },  
    v1: { /* A Constraint Object */ },  
}
```

The node names in a query path ( $v_0, v_1, \dots$ ) serve as node identifiers that can be referenced in the query object; the edge names ( $e_0, e_1, \dots$ ) in the path represent the types of the corresponding edges. We can use an asterisk \* as a node or edge name (except for the starting node, which must be given) to declare that there are no constraints on such an element. For example, a query path `/v0/*/v1/e1/*/` retrieves paths from the graph without restricting the type of the edge ( $v_0, v_1$ ). Meanwhile, the query does not have constraints on the destination (the last node) of the path either.

When a path contains just one node, say  $v_0$ , the query will simply return all entities that satisfy the constraints. A constraint object applied to the starting node is called a *Starting Query Object*, whose specification is given as follows.

```
/* Starting Query Object:  
Specifies constraints on the starting node  
*/  
{  
    /* "match" operator searches for entities with the specified properties.  
       All properties defined in the graph schema be queried such as "Name" and "NormalizedTitle".  
    */  
    "match": {  
        "Name" : "some name",  
        ...  
    },  
    /* "id" operator directly specifies the IDs of the starting nodes.  
       When it is present, the "match" operator is ignored.  
    */  
    "id": [ id1, id2, id3... ],  
    /* "type" operator limits results to a certain type of entities,  
       for example, "Author" or "Paper".  
    */  
    "type": "Author",  
    /* "select" operator pulls properties from the database and  
       returns them to the client.  
    */  
    "select": ["PaperRank", ...]  
}
```

When a path contains more than just a starting node, the query processor will perform a graph traversal following the specified path pattern. When it arrives at a node, the user-specified traversal actions will be triggered, that is, whether to stop at the current node and return or to continue to explore the graph. When no traversal action is specified, default actions will be taken. For an intermediate node, the default action is to continue to explore the

graph. For the last node of a path, the default action is to stop and return. A constraint object that specifies traversal actions is called a *Traversal Action Object*. Its specification is given as follows:

```

/* Traversal Action Object:
   Specifies graph traversal actions on a node (except for the starting node).
*/
{
  /* Set the continue condition here. */
  continue: {
    /* simple property exact match */
    "property_key" : "property_value",
    /* Advanced query operators */
    /* -- Numerical comparisons */
    "property_key_2" : { "gt" /* or simply ">" */ : 123 },
    /* -- Substring query */
    "property_key_3" : { "substring" : "456" },
    /* -- CellID query */
    "id": [ 111, 222, 333... ],
    /* -- Entity type query */
    "type": "cell_type",
    /* -- Property existance query */
    "has" : "property_key_4",
    /* -- Logical operators */
    /* ---- Note that, by default the operators are combined with AND semantics */

    /* -- OR operator */
    "or": {
      /* Query operators... */
    },
    /* -- NOT operator */
    "not": {
      /* Query operators... */
    }
  },
  /* Set the return condition here. */
  return: {
    /* Same operators as the continue object */
  },
  /* The selected properties to return. */
  select: ["property_key_1", ...]
}

```

The POST body of a *json* search query should contain at least a *path* pattern. Traverse action objects are optional. Here are two examples.

```
{
  "path": "/series/ConferenceInstanceIDs/conference/FieldOfStudyIDs/field",
  "series": {
    "type": "ConferenceSeries",
    "FullName": "graph",
    "select": [ "FullName", "ShortName" ]
  },
  "conference": {
    "type": "ConferenceInstance",
    "select": [ "FullName" ]
  },
  "field": {
    "type": "FieldOfStudy",
    "select": [ "Name" ],
    "return": { "Name": { "substring" : "World Wide Web" } }
  }
}
```

```
{  
  "path": "/author/PaperIDs/paper",  
  "author": {  
    "type": "Author",  
    "select": [ "DisplayAuthorName" ],  
    "match": { "Name": "leslie lamport" }  
  },  
  "paper": {  
    "type": "Paper",  
    "select": [ "OriginalTitle", "CitationCount" ],  
    "return": { "CitationCount": { "gt": 100 } }  
  }  
}
```

# Bing Autosuggest API

4/12/2017 • 1 min to read • [Edit Online](#)

The Bing Autosuggest API lets partners send a partial search query to Bing and get back a list of suggested queries that other users have searched on (overview on [MSDN](#)). In addition to including searches that other users have made, the list may include suggestions based on user intent. For example, if the query string is "weather in Lo", the list will include relevant weather suggestions.

Typically, you use this API to support an auto-suggest search box feature. For example, as the user types a query into the search box, you would call this API to populate a drop-down list of suggested query strings. If the user selects a query from the list, you would either send the user to the Bing search results page for the selected query or call the [Web Search API](#) to get the search results and display the results yourself.

To get started, read our [Getting Started](#) guide, which describes how you can obtain your own subscription keys and start making calls to the API. If you already have a subscription, try our API Testing Console [API Testing Console](#) where you can easily craft API requests in a sandbox environment.

For information that shows you how to use the Autosuggest API, see [Autosuggest Guide](#).

For information about the programming elements that you'd use to request and consume the search results, see [Autosuggest Reference](#).

For additional guide and reference content that is common to all Bing APIs, such as Paging Results and Error Codes, see [Shared Guides](#) and [Shared Reference](#).

# Bing Search API Use and Display Requirements

4/12/2017 • 7 min to read • [Edit Online](#)

These use and display requirements apply to your implementation of the content and associated information (for example, relationships, metadata and other signals) available through calls to the Bing Web Search, Image Search, Video Search, and News Search APIs, Bing Spell Check and Bing Autosuggest APIs. Implementation details related to these requirements can be found in documentation for specific features and results.

## 1. BING SPELL CHECK API and BING AUTOSUGGEST API. You must not:

- copy, store, or cache any data you receive from the Bing Spell Check or Bing Autosuggest APIs; or use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.
- use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

## 2. BING WEB SEARCH, IMAGE SEARCH, NEWS SEARCH and VIDEO SEARCH APIs (the "Search APIs"):

**Definitions.** The following definitions apply in Sections 2 through 7 of these use and display requirements:

- "answer" refers to a category of results returned in a response. For example, a response from the Bing Web Search API may include answers in the categories of webpage results, image, video, and news;
- "response" means any and all answers and associated data received in response to a single call to a Search API;
- "result" refers to an item of information in an answer. For example, the set of data connected with a single news article is a result in a news answer.

**Internet search experience.** All data returned in responses may only be used in Internet search experiences. An Internet search experience means the content displayed, as applicable:

- is relevant and responsive to the end user's direct query or other indication of the user's search interest and intent (e.g., user-indicated search query);
- helps users find and navigate to the sources of data (e.g., the provided URLs are implemented as hyperlinks so the content or attribution is a clickable link conspicuously displayed with the data);
- includes multiple results for the end user to select from (e.g., several results from the news answer are displayed, or all results if fewer than several are returned);
- is limited to an amount appropriate to serve the search purpose (e.g., image thumbnails are thumbnail-sized in proportion to the user's display);
- includes visible indication to the end user that the content is Internet search results (e.g., a statement that the content is "From the web"); and
- includes any other combination of measures appropriate to ensure your use of data received from the Search APIs does not violate any applicable laws or third party rights. Please consult your legal advisors to determine what measures may be appropriate.

The only exception to the internet search experience requirement is for URL discovery as described in Section 7 (Non-display URL discovery) below.

**General.** You must not:

- copy, store, or cache any data from responses (except retention to the extent permitted by the "Continuity of Service" section below);
- modify content of results (other than to reformat them in a way that does not violate any other requirement);
- omit attribution and URLs associated with result content;
- re-order (including by omission) results displayed in an answer when an order or ranking is provided;
- display other content within any part of a response in a way that would lead an end user to believe that the other content is part of the response;
- display advertising that is not provided by Microsoft on any page that displays any part of a response;
- use data received from the Search APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

### 3. Advertising.

Advertising (whether provided by Microsoft or another provider) must not be displayed with responses (i) from the Image, News or Video Search APIs; or (ii) that are filtered or limited primarily (or solely) to image, news and/or video results from other Search APIs.

### 4. Branding.

You will attribute each response (or portion of a response) displayed to Microsoft as described in <https://go.microsoft.com/fwlink/?linkid=833278>, unless Microsoft specifies otherwise in writing for your particular use.

### 5. Transferring Responses.

If you enable a user to transfer a response to another user, such as through a messaging app or social media posting, the following apply:

- Transferred responses must:
  - Consist of content that is unmodified from the content of the responses displayed to the transferring user (formatting changes are permissible);
  - Not include any data in metadata form;
  - Display language indicating the response was obtained through an Internet search experience powered by Bing (e.g., "Powered by Bing," "Learn more about this image on Bing," or "Explore more about this image on Bing" or through the use of the Bing logo);
  - Prominently display the full query used to generate the response; and
  - Include a prominent link or similar attribution to the underlying source of the response, either directly or through bing.com or m.bing.com.
- You may not automate the transfer of responses. A transfer must be initiated by a user action clearly evidencing an intent to transfer a response.
- You may only enable a user to transfer responses obtained as a result of the transferring user's query.

## 6. Continuity of Service.

You must not copy, store or cache any data from responses. However, to enable continuity of service access and data rendering, you may retain results solely under the following conditions:

**Device.** You may enable an end user to retain results on a device for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results, provided that retained results may be used only:

- to enable the end user to access results previously returned to that end user on that device (e.g., in case of service interruption); or
- to store results returned for your proactive query personalized in anticipation of the end user's needs based on that end user's signals (e.g., in case of anticipated service interruption).

**Server.** You may retain results specific to a single end user securely on a server you control and display the retained results only:

- to enable the end user to access a historical report of results previously returned to that user in your solution, provided that the results may not be (i) retained for more than 21 days from the time of the end user's initial query and (ii) displayed in response to an end user's new or repeated query; or
- to store results returned for your proactive query personalized in anticipation of an end user's needs based on that end user's signals for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results.

Whenever retained, results for a specific user cannot be commingled with results for another user, i.e., the results of each user must be retained and delivered separately.

**General.** For all presentation of retained results, you must:

- include a clear, visible notice of the time the query was sent,
- present the user a button or similar means to re-query and obtain updated results,
- retain the Bing branding in the presentation of the results, and
- delete (and refresh with a new query if needed) the stored results within the timeframes specified.

## 7. Non-display URL discovery.

You may only use search responses in a non-internet search experience for the sole purpose of discovering URLs of sources of information responsive to a query from your user or customer. You may copy such URLs in a report or similar response you provide (i) only to that user or customer, in response to the particular query and (ii) which includes significant additional valuable content relevant to the query. The requirements in sections 2 through 6 of these use and display requirements do not apply to this non-display use, except:

- You shall not cache, copy or store any data or content from, or derived from, the search response, other than the limited URL copying described above;
- You must ensure your use of data (including the URLs) received from the Search APIs does not violate any applicable laws or third party rights; and
- You shall not use the data (including the URLs) received from the Search APIs as part of any search index or machine learning or similar algorithmic activity to create train, evaluate, or improve services which you or third parties may offer.

# Bing Image Search API

4/12/2017 • 1 min to read • [Edit Online](#)

The Bing Image Search API provides a similar (but not exact) experience to Bing.com/Images (overview on [MSDN](#)). The Image Search API lets partners send a search query to Bing and get back a list of relevant images.

To get started, read our [Getting Started](#) guide, which describes how you can obtain your own subscription keys and start making calls to the API. If you already have a subscription, try our API Testing Console [API Testing Console](#) where you can easily craft API requests in a sandbox environment.

For information that shows you how to use the Image Search API, see [Image Search API Guide](#) on MSDN. You can also refer to a bot sample [here](#) that uses this API to find visually similar products.

For information about the programming elements that you'd use to request and consume the search results, see [Image Search API Reference](#).

For additional guide and reference content that is common to all Bing APIs, such as Paging Results and Error Codes, see [Shared Guides](#) and [Shared Reference](#).

# Bing Search API Use and Display Requirements

4/12/2017 • 7 min to read • [Edit Online](#)

These use and display requirements apply to your implementation of the content and associated information (for example, relationships, metadata and other signals) available through calls to the Bing Web Search, Image Search, Video Search, and News Search APIs, Bing Spell Check and Bing Autosuggest APIs. Implementation details related to these requirements can be found in documentation for specific features and results.

## 1. BING SPELL CHECK API and BING AUTOSUGGEST API. You must not:

- copy, store, or cache any data you receive from the Bing Spell Check or Bing Autosuggest APIs; or use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.
- use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

## 2. BING WEB SEARCH, IMAGE SEARCH, NEWS SEARCH and VIDEO SEARCH APIs (the "Search APIs"):

**Definitions.** The following definitions apply in Sections 2 through 7 of these use and display requirements:

- "answer" refers to a category of results returned in a response. For example, a response from the Bing Web Search API may include answers in the categories of webpage results, image, video, and news;
- "response" means any and all answers and associated data received in response to a single call to a Search API;
- "result" refers to an item of information in an answer. For example, the set of data connected with a single news article is a result in a news answer.

**Internet search experience.** All data returned in responses may only be used in Internet search experiences. An Internet search experience means the content displayed, as applicable:

- is relevant and responsive to the end user's direct query or other indication of the user's search interest and intent (e.g., user-indicated search query);
- helps users find and navigate to the sources of data (e.g., the provided URLs are implemented as hyperlinks so the content or attribution is a clickable link conspicuously displayed with the data);
- includes multiple results for the end user to select from (e.g., several results from the news answer are displayed, or all results if fewer than several are returned);
- is limited to an amount appropriate to serve the search purpose (e.g., image thumbnails are thumbnail-sized in proportion to the user's display);
- includes visible indication to the end user that the content is Internet search results (e.g., a statement that the content is "From the web"); and
- includes any other combination of measures appropriate to ensure your use of data received from the Search APIs does not violate any applicable laws or third party rights. Please consult your legal advisors to determine what measures may be appropriate.

The only exception to the internet search experience requirement is for URL discovery as described in Section 7 (Non-display URL discovery) below.

**General.** You must not:

- copy, store, or cache any data from responses (except retention to the extent permitted by the "Continuity of Service" section below);
- modify content of results (other than to reformat them in a way that does not violate any other requirement);
- omit attribution and URLs associated with result content;
- re-order (including by omission) results displayed in an answer when an order or ranking is provided;
- display other content within any part of a response in a way that would lead an end user to believe that the other content is part of the response;
- display advertising that is not provided by Microsoft on any page that displays any part of a response;
- use data received from the Search APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

### 3. Advertising.

Advertising (whether provided by Microsoft or another provider) must not be displayed with responses (i) from the Image, News or Video Search APIs; or (ii) that are filtered or limited primarily (or solely) to image, news and/or video results from other Search APIs.

### 4. Branding.

You will attribute each response (or portion of a response) displayed to Microsoft as described in <https://go.microsoft.com/fwlink/?linkid=833278>, unless Microsoft specifies otherwise in writing for your particular use.

### 5. Transferring Responses.

If you enable a user to transfer a response to another user, such as through a messaging app or social media posting, the following apply:

- Transferred responses must:
  - Consist of content that is unmodified from the content of the responses displayed to the transferring user (formatting changes are permissible);
  - Not include any data in metadata form;
  - Display language indicating the response was obtained through an Internet search experience powered by Bing (e.g., "Powered by Bing," "Learn more about this image on Bing," or "Explore more about this image on Bing" or through the use of the Bing logo);
  - Prominently display the full query used to generate the response; and
  - Include a prominent link or similar attribution to the underlying source of the response, either directly or through bing.com or m.bing.com.
- You may not automate the transfer of responses. A transfer must be initiated by a user action clearly evidencing an intent to transfer a response.
- You may only enable a user to transfer responses obtained as a result of the transferring user's query.

## 6. Continuity of Service.

You must not copy, store or cache any data from responses. However, to enable continuity of service access and data rendering, you may retain results solely under the following conditions:

**Device.** You may enable an end user to retain results on a device for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results, provided that retained results may be used only:

- to enable the end user to access results previously returned to that end user on that device (e.g., in case of service interruption); or
- to store results returned for your proactive query personalized in anticipation of the end user's needs based on that end user's signals (e.g., in case of anticipated service interruption).

**Server.** You may retain results specific to a single end user securely on a server you control and display the retained results only:

- to enable the end user to access a historical report of results previously returned to that user in your solution, provided that the results may not be (i) retained for more than 21 days from the time of the end user's initial query and (ii) displayed in response to an end user's new or repeated query; or
- to store results returned for your proactive query personalized in anticipation of an end user's needs based on that end user's signals for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results.

Whenever retained, results for a specific user cannot be commingled with results for another user, i.e., the results of each user must be retained and delivered separately.

**General.** For all presentation of retained results, you must:

- include a clear, visible notice of the time the query was sent,
- present the user a button or similar means to re-query and obtain updated results,
- retain the Bing branding in the presentation of the results, and
- delete (and refresh with a new query if needed) the stored results within the timeframes specified.

## 7. Non-display URL discovery.

You may only use search responses in a non-internet search experience for the sole purpose of discovering URLs of sources of information responsive to a query from your user or customer. You may copy such URLs in a report or similar response you provide (i) only to that user or customer, in response to the particular query and (ii) which includes significant additional valuable content relevant to the query. The requirements in sections 2 through 6 of these use and display requirements do not apply to this non-display use, except:

- You shall not cache, copy or store any data or content from, or derived from, the search response, other than the limited URL copying described above;
- You must ensure your use of data (including the URLs) received from the Search APIs does not violate any applicable laws or third party rights; and
- You shall not use the data (including the URLs) received from the Search APIs as part of any search index or machine learning or similar algorithmic activity to create train, evaluate, or improve services which you or third parties may offer.

# Bing News Search API

4/12/2017 • 1 min to read • [Edit Online](#)

The Bing News Search API provides a similar (but not exact) experience to Bing.com/News (overview on [MSDN](#)). The Bing News Search API lets partners send a search query to Bing and get back a list of relevant news articles.

To get started, read our [Getting Started](#) guide, which describes how you can obtain your own subscription keys and start making calls to the API. If you already have a subscription, try our API Testing Console [API Testing Console](#) where you can easily craft API requests in a sandbox environment.

For information that shows you how to use the API, see [News Guide](#).

For information about the programming elements that you'd use to request and consume the search results, see [News Reference](#).

For additional guide and reference content that is common to all Bing APIs, such as Paging Results and Error Codes, see [Shared Guides](#) and [Shared Reference](#).

# Bing Search API Use and Display Requirements

4/12/2017 • 7 min to read • [Edit Online](#)

These use and display requirements apply to your implementation of the content and associated information (for example, relationships, metadata and other signals) available through calls to the Bing Web Search, Image Search, Video Search, and News Search APIs, Bing Spell Check and Bing Autosuggest APIs. Implementation details related to these requirements can be found in documentation for specific features and results.

## 1. BING SPELL CHECK API and BING AUTOSUGGEST API. You must not:

- copy, store, or cache any data you receive from the Bing Spell Check or Bing Autosuggest APIs; or use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.
- use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

## 2. BING WEB SEARCH, IMAGE SEARCH, NEWS SEARCH and VIDEO SEARCH APIs (the "Search APIs"):

**Definitions.** The following definitions apply in Sections 2 through 7 of these use and display requirements:

- "answer" refers to a category of results returned in a response. For example, a response from the Bing Web Search API may include answers in the categories of webpage results, image, video, and news;
- "response" means any and all answers and associated data received in response to a single call to a Search API;
- "result" refers to an item of information in an answer. For example, the set of data connected with a single news article is a result in a news answer.

**Internet search experience.** All data returned in responses may only be used in Internet search experiences. An Internet search experience means the content displayed, as applicable:

- is relevant and responsive to the end user's direct query or other indication of the user's search interest and intent (e.g., user-indicated search query);
- helps users find and navigate to the sources of data (e.g., the provided URLs are implemented as hyperlinks so the content or attribution is a clickable link conspicuously displayed with the data);
- includes multiple results for the end user to select from (e.g., several results from the news answer are displayed, or all results if fewer than several are returned);
- is limited to an amount appropriate to serve the search purpose (e.g., image thumbnails are thumbnail-sized in proportion to the user's display);
- includes visible indication to the end user that the content is Internet search results (e.g., a statement that the content is "From the web"); and
- includes any other combination of measures appropriate to ensure your use of data received from the Search APIs does not violate any applicable laws or third party rights. Please consult your legal advisors to determine what measures may be appropriate.

The only exception to the internet search experience requirement is for URL discovery as described in Section 7 (Non-display URL discovery) below.

**General.** You must not:

- copy, store, or cache any data from responses (except retention to the extent permitted by the "Continuity of Service" section below);
- modify content of results (other than to reformat them in a way that does not violate any other requirement);
- omit attribution and URLs associated with result content;
- re-order (including by omission) results displayed in an answer when an order or ranking is provided;
- display other content within any part of a response in a way that would lead an end user to believe that the other content is part of the response;
- display advertising that is not provided by Microsoft on any page that displays any part of a response;
- use data received from the Search APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

### 3. Advertising.

Advertising (whether provided by Microsoft or another provider) must not be displayed with responses (i) from the Image, News or Video Search APIs; or (ii) that are filtered or limited primarily (or solely) to image, news and/or video results from other Search APIs.

### 4. Branding.

You will attribute each response (or portion of a response) displayed to Microsoft as described in <https://go.microsoft.com/fwlink/?linkid=833278>, unless Microsoft specifies otherwise in writing for your particular use.

### 5. Transferring Responses.

If you enable a user to transfer a response to another user, such as through a messaging app or social media posting, the following apply:

- Transferred responses must:
  - Consist of content that is unmodified from the content of the responses displayed to the transferring user (formatting changes are permissible);
  - Not include any data in metadata form;
  - Display language indicating the response was obtained through an Internet search experience powered by Bing (e.g., "Powered by Bing," "Learn more about this image on Bing," or "Explore more about this image on Bing" or through the use of the Bing logo);
  - Prominently display the full query used to generate the response; and
  - Include a prominent link or similar attribution to the underlying source of the response, either directly or through bing.com or m.bing.com.
- You may not automate the transfer of responses. A transfer must be initiated by a user action clearly evidencing an intent to transfer a response.
- You may only enable a user to transfer responses obtained as a result of the transferring user's query.

## 6. Continuity of Service.

You must not copy, store or cache any data from responses. However, to enable continuity of service access and data rendering, you may retain results solely under the following conditions:

**Device.** You may enable an end user to retain results on a device for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results, provided that retained results may be used only:

- to enable the end user to access results previously returned to that end user on that device (e.g., in case of service interruption); or
- to store results returned for your proactive query personalized in anticipation of the end user's needs based on that end user's signals (e.g., in case of anticipated service interruption).

**Server.** You may retain results specific to a single end user securely on a server you control and display the retained results only:

- to enable the end user to access a historical report of results previously returned to that user in your solution, provided that the results may not be (i) retained for more than 21 days from the time of the end user's initial query and (ii) displayed in response to an end user's new or repeated query; or
- to store results returned for your proactive query personalized in anticipation of an end user's needs based on that end user's signals for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results.

Whenever retained, results for a specific user cannot be commingled with results for another user, i.e., the results of each user must be retained and delivered separately.

**General.** For all presentation of retained results, you must:

- include a clear, visible notice of the time the query was sent,
- present the user a button or similar means to re-query and obtain updated results,
- retain the Bing branding in the presentation of the results, and
- delete (and refresh with a new query if needed) the stored results within the timeframes specified.

## 7. Non-display URL discovery.

You may only use search responses in a non-internet search experience for the sole purpose of discovering URLs of sources of information responsive to a query from your user or customer. You may copy such URLs in a report or similar response you provide (i) only to that user or customer, in response to the particular query and (ii) which includes significant additional valuable content relevant to the query. The requirements in sections 2 through 6 of these use and display requirements do not apply to this non-display use, except:

- You shall not cache, copy or store any data or content from, or derived from, the search response, other than the limited URL copying described above;
- You must ensure your use of data (including the URLs) received from the Search APIs does not violate any applicable laws or third party rights; and
- You shall not use the data (including the URLs) received from the Search APIs as part of any search index or machine learning or similar algorithmic activity to create train, evaluate, or improve services which you or third parties may offer.

# Bing Speech API Overview

4/21/2017 • 2 min to read • [Edit Online](#)

Welcome to Bing Speech API, a cloud-based API that provides advanced algorithms to process spoken language. With Bing Speech API you can add speech driven actions to your apps, including real-time interaction with the user.

Bing Speech API has two components:

- [Speech To Text API](#): For apps converting spoken words to text.
- [Text To Speech API](#): For apps converting text into audio that can be played back to the user.

Bing Speech APIs and libraries enables speech capabilities on all internet-connected devices. Every major platform including Android, iOS, Windows, and 3rd party IoT devices are supported. It offers industry-leading speech-to-text, text-to-speech, and language understanding capabilities delivered through the cloud.

Microsoft uses Bing Speech API for Windows applications like [Cortana](#) and [Skype Translator](#) as well as Android applications like [Bing Torque](#) for Android Wear and Android Phone.

## Speech Recognition API

Bing Speech Recognition API provides the ability to convert spoken audio to text by sending audio to Microsoft's servers in the cloud. Developers have a choice of using the REST API, Client Library or the Service Library.

### Speech Recognition - REST API versus Client Library versus Service Library

- Using the REST API means getting only one reco result back with no partial results. Documentation for the REST API can be found [here](#) and code samples [here](#).
- Using the client library allows for real-time streaming, meaning that as audio is being sent or spoken to the server, partial recognition results are returned at the same time. Real-time streaming is supported on Android, iOS, and Windows. The client library also supports speech intent recognition in addition to returning recognized text from audio inputs. Structured information about the speech to apps that parse the intent of the speaker can also be retrieved to drive further actions. Models trained by [Project LUIS](#) service are used to generate the intent. To use intent, you will need to train a model after getting an AppID and a Secret. Once you have a trained model, you can use the Speech Recognition API for intent parsing on reco results via the "WithIntent" clients.
- Using the service library allows for real-time streaming audio from a service to the speech cloud allowing for partial results. Service Library is supported for Windows.

### Supported languages

Locales supported by the Speech Recognition API include:

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
ar-EG*	en-NZ	it-IT	ru-RU
ca-ES	en-US	ja-JP	sv-SE
da-DK	es-ES	ko-KR	zh-CN
de-DE	es-MX	nb-NO	zh-HK
en-AU	fi-FI	nl-NL	zh-TW

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
en-CA	fr-CA	pl-PL	
en-GB	fr-FR	pt-BR	
en-IN	hi-IN	pt-PT	

\*ar-EG supports Modern Standard Arabic (MSA)

## Text To Speech API

When applications need to “talk” back to their users, this API can be used to convert text generated by the app into audio that can be played back to the user. Text to speech conversion (TTS) is done via a REST API. The TTS demo can be found on the [Speech landing page](#), documentation for the REST API can be found [here](#), and code samples [here](#).

### Supported languages

Locales supported by Text to Speech API include:

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
ar-EG*	es-ES	ko-KR
de-DE	es-MX	pt-BR
en-AU	fr-CA	ru-RU
en-CA	fr-FR	zh-CN
en-GB	hi-IN	zh-HK
en-IN	it-IT	zh-TW
en-US	ja-JP	

\*ar-EG supports Modern Standard Arabic (MSA)

# Get Started with Bing Speech Recognition API in Objective C on iOS

4/12/2017 • 7 min to read • [Edit Online](#)

With Bing Speech Recognition API you can develop iOS applications that leverage Microsoft cloud servers to convert spoken audio to text. The API supports real-time streaming, so your application can simultaneously and asynchronously receive partial recognition results at the same time it is sending audio to the service.

This article uses a sample application to demonstrate the basics of getting started with the Bing Speech Recognition API to develop an iOS application. For a complete API reference, see [Speech SDK Client Library Reference](#).

## Prerequisites

### Platform requirements

Make sure Mac XCode IDE is installed.

### Get the client library and example

You may download the Speech API client library and example for iOS through <https://www.microsoft.com/cognitive-services/en-us/speech-api>. The downloaded zip file needs to be extracted to a folder of your choice. Install the .pkg file on your Mac. The .pkg file will install onto your Mac hard drive in the root (or personal) Documents directory under **SpeechSDK**. Inside the folder there is both a fully buildable example and an SDK library. The buildable example can be found in the **samples\SpeechRecognitionServerExample** directory and the library can be found at the **SpeechSDK\SpeechSDK.framework**.

### Subscribe to Speech API and get a free trial subscription key

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

## Step 1: Install the Example Application and Create the Application Framework

Open Xcode IDE. You have two options, building the example application or building your own application.

If you want build and run the **example application**, the project is embedded [here](#) at **samples\SpeechRecognitionServerExample** and can be opened in XCode.

- You will need to paste your subscription key into the file "**settings.plist**" which can be found in the **samples** folder under **SpeechRecognitionServerExample**. (You may ignore the LUIS values if you don't want to use "Intent" right now.)

If you want to **build your own application**, continue on with these instructions.

1. Create a new application project.
2. With the items you downloaded from the SDK, do the following:
  - a) Click on the project in the file navigator on the left. Then click on the project or target in the editor that appears. Click on "**Build Settings**", then change from "**Basic**" to "**All**".
  - b) Inside the directory to which you unpacked the SDK, you will see the directory, **SpeechSDK/SpeechSDK.framework/Headers**. Add an "**Include Search Path**" to include the **Headers**

directory.

- c) Inside the directory to which you unpacked the SDK, you will see the directory, **SpeechSDK**. Add a "Framework Search Path" to include the **SpeechSDK** directory.
- d) Click on the project in the file navigator on the left. Then click on the project or target in the editor that appears. Click on "**General**".
- e) Inside the directory to which you unpacked the SDK, you will see the directory, **SpeechSDK/SpeechSDK.framework**. Add **SpeechSDK/SpeechSDK.framework** as a "**Linked Frameworks and Libraries**" via the "**Add Other...**" button found after you click on "+".
- f) Also add **SpeechSDK.framework** as an "**Embedded Binary**" framework.
- g) Note that inside the directory to which you unpacked the SDK in directory **SpeechSDK\Samples\SpeechRecognitionServerExample** there is a XCode buildable example so you can see these settings in action.

## Step 2: Build the Application / Example Code

Open [ViewController.mm](#) in a new window or find **ViewController.mm** in the downloaded file under **samples\SpeechRecognitionServiceExample**. You will need the **Speech API primary subscription key**. The below code snippet shows where to use the key. (You may ignore the LUIS values if you don't want to use "Intent" right now.)

```

{
    NSString* language = @"en-us";

    NSString* path = [[NSBundle mainBundle] pathForResource:@"settings" ofType:@"plist"];
    NSDictionary* settings = [[NSDictionary alloc] initWithContentsOfFile:path];

    NSString* primaryOrSecondaryKey = [settings objectForKey:(@"primaryKey")];
    NSString* luisAppID = [settings objectForKey:(@"luisAppID")];
    NSString* luisSubscriptionID = [settings objectForKey:(@"luisSubscriptionID")];

    if (isMicrophoneReco) {
        if (!isIntent) {
            micClient = [SpeechRecognitionServiceFactory createMicrophoneClient:(recoMode)
                withLanguage:(language)
                withKey:(primaryOrSecondaryKey)
                withProtocol:(self)];
        }
        else {
            MicrophoneRecognitionClientWithIntent* micIntentClient;
            micIntentClient = [SpeechRecognitionServiceFactory createMicrophoneClientWithIntent:(language)
                withKey:
                (primaryOrSecondaryKey)
                withLUISAppID:(luisAppID)
                withLUISSecret:
                (luisSubscriptionID)
                withProtocol:(self)];
            micClient = micIntentClient;
        }
    }
    else {
        if (!isIntent) {
            dataClient = [SpeechRecognitionServiceFactory createDataClient:(recoMode)
                withLanguage:(language)
                withKey:(primaryOrSecondaryKey)
                withProtocol:(self)];
        }
        else {
            DataRecognitionClientWithIntent* dataIntentClient;
            dataIntentClient = [SpeechRecognitionServiceFactory createDataClientWithIntent:(language)
                withKey:
                (primaryOrSecondaryKey)
                withLUISAppID:(luisAppID)
                withLUISSecret:
                (luisSubscriptionID)
                withProtocol:(self)];
            dataClient = dataIntentClient;
        }
    }
}

```

#### Create a Client

Once your **primaryKey** has been pasted into the example, you can use the **SpeechRecognitionServiceFactory** to create a client of your liking. For example, you can create a client consisting of:

- **DataRecognitionClient:** Speech recognition with PCM data (for example from a file or audio source). The data is broken up into buffers and each buffer is sent to the Speech Recognition Service. No modification is done to the buffers, so the user can apply their own Silence Detection if desired. If the data is provided from wave files, you can send data from the file right to the server. If you have raw data, for example audio coming over Bluetooth, then you first send a format header to the server followed by the data.
- **MicrophoneRecognitionClient:** Speech recognition with audio coming from the microphone. Make sure the microphone is turned on and data from the microphone is sent to the Speech Recognition Service. A built-in "Silence Detector" is applied to the microphone data before it is sent to the recognition service.

- **“WithIntent” Clients:** Use “WithIntent” if you want the server to return additional structured information about the speech to be used by apps to parse the intent of the speaker and drive further actions by the app. To use Intent, you will need to train a model and get an AppID and a Secret. See project [LUIS](#) for details.

#### Select a Language

When you use the SpeechRecognitionServiceFactory to create the Client, you must select a language.

Supported locales include:

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
de-DE	zh-TW	zh-HK	ru-RU
es-ES	ja-JP	ar-EG*	da-DK
en-GB	en-IN	fi-FI	nl-NL
en-US	pt-BR	pt-PT	ca-ES
fr-FR	ko-KR	en-NZ	nb-NO
it-IT	fr-CA	pl-PL	es-MX
zh-CN	en-AU	en-CA	sv-SE

\*ar-EG supports Modern Standard Arabic (MSA)

#### Select a Recognition Mode

You also need to provide the recognition mode.

- **ShortPhrase mode:** An utterance up to 15 seconds long. As data is sent to the service, the client will receive multiple partial results and one final multiple n-best choice result.
- **LongDictation mode:** An utterance up to 2 minutes long. As data is sent to the service, the client will receive multiple partial results and multiple final results, based on where the server identifies sentence pauses.

#### Attach Event Handlers

You can attach various event handlers to the client you created.

- **Partial Results Events:** This event gets called every time the Speech Recognition Service predicts what you might be saying – even before you finish speaking (if you are using the Microphone Client) or have finished sending data (if you are using the Data Client).
- **Error Events:** Called when the service detects an Error.
- **Intent Events:** Called on “WithIntent” clients (only in ShortPhrase mode) after the final reco result has been parsed into a structured JSON intent.
- **Result Events:**
  - **In ShortPhrase mode**, this event is called and returns n-best results after you finish speaking.
  - **In LongDictation mode**, the event handler is called multiple times, based on where the service identifies sentence pauses.
  - **For each of the n-best choices**, a confidence value and a few different forms of the recognized text are returned:
    - **LexicalForm:** This form is optimal for use by applications that need the raw, unprocessed

speech recognition result.

- **DisplayText:** The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, for example "d\*\*\*". This form is optimal for use by applications that display the speech recognition results to users.
- **Inverse Text Normalization (ITN):** An example of ITN is converting result text from "go to fourth street" to "go to 4th St". This form is optimal for use by applications that display the speech recognition results to users.
- **InverseTextNormalizationResult:** Inverse text normalization (ITN) converts phrases like "one two three four" to a normalized form such as "1234". Another example is converting result text from "go to fourth street" to "go to 4th St". This form is optimal for use by applications that interpret the speech recognition results as commands or perform queries based on the recognized text.
- **MaskedInverseTextNormalizationResult:** The recognized phrase with inverse text normalization and profanity masking applied, but no capitalization or punctuation. Profanity is masked with asterisks after the initial character, e.g. "d\*\*\*". This form is optimal for use by applications that display the speech recognition results to users. Inverse Text Normalization (ITN) has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that use the unmasked ITN results, but also need to display the command or query to users.

## Step 3: Run the Example Application

Run the application with the chosen clients, recognition modes and event handlers.

## Related Topics

- [Get started with Bing Speech Recognition and/or intent in Java on Android](#)
- [Get started with Bing Speech API in JavaScript](#)
- [Get started with Bing Speech API in cURL](#)

# Get Started with Bing Speech API in cURL

4/12/2017 • 1 min to read • [Edit Online](#)

Exercise Bing Speech Recognition API using cURL to convert spoken audio to text by sending audio to Microsoft's servers in the cloud. The example below is **bash** commands that demonstrates the use of Microsoft Cognitive Services (formerly Project Oxford) Speech To Text API using **cURL**.

The Speech Recognition web example demonstrates the following features using a wav file or external microphone input:

- Access token generation
- Short-form recognition This example assumes that **cURL** is available in your bash environment.

## Prerequisites

- **Platform requirements**

The below example has been developed in **bash**. (Also works in git bash/zsh/etc)

- **Subscribe to Speech API and get a free trial subscription key**

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

## Step 1: Generate an Access Token

1. Replace **your\_subscription\_key** with your own subscription key and run the command in **bash**.

```
curl -v -X POST "https://api.cognitive.microsoft.com/sts/v1.0/issueToken" -H "Content-type: application/x-www-form-urlencoded" -H "Content-Length: 0" -H "Ocp-Apim-Subscription-Key: your_subscription_key"
```

2. The response is a string with the JSON Web Token (JWT) access token. [JWT access token](#)

## Step 2: Upload the Audio Binary

1. Replace **your\_instance\_id**, **your\_request\_id**, **your\_locale**, **your\_device\_os** in accordance to your own application
2. Replace **your\_access\_token** with the JWT access token retrieved from [Step 1](#)
3. Replace **your\_wave\_file** with the actual wave file
4. Run the command in **bash**

```
curl -v -X POST "https://speech.platform.bing.com/recognize?scenarios=smd&appid=D4D52672-91D7-4C74-8AD8-42B1D98141A5&locale=your_locale&device.os=your_device_os&version=3.0&format=json&instanceid=your_instance_id&requestid=your_request_id" -H 'Authorization: Bearer your_access_token' -H 'Content-type: audio/wav; codec="audio/pcm"; samplerate=16000' --data-binary @your_wave_file
```

5. Parse the Successful recognition response or Error response

## Related Topics

- [Get Started with Bing Speech Recognition in C Sharp for .Net on Windows Desktop](#)
- [Get Started with Bing Speech Recognition in Java on Android](#)
- [Get Started with Bing Speech Recognition in JavaScript](#)
- [Get Started with Bing Speech Recognition in Objective C on iOS](#)

For questions, feedback, or suggestions about Microsoft Cognitive Services, feel free to reach out to us directly.

- Cognitive Services [UserVoice Forum](#)

## License

All Microsoft Cognitive Services SDKs and samples are licensed with the MIT License. For more details, see [LICENSE](#).

# Getting Started with Bing Speech Recognition in C# for .Net Windows

4/12/2017 • 5 min to read • [Edit Online](#)

Develop a basic Windows application that uses Bing Speech Recognition API to convert spoken audio to text by sending audio to Microsoft's servers in the cloud. Using the Client Library allows for real-time streaming, which means that at the same time your client application sends audio to the service, it simultaneously and asynchronously receives partial recognition results back. This page describes use of the Client Library, which currently supports speech in seven languages, the example below defaults to American English, "en-US". For client library API reference, see [Microsoft Bing Speech SDK](#).

## Prerequisites

### Platform Requirements

The below example has been developed for Windows 8+ and .NET Framework 4.5+ using [Visual Studio 2015, Community Edition](#).

### Get the Client Library and Example

You may download the Speech API client library and example through [SDK](#). The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder.

### Subscribe to Speech API and Get a Free Trial Subscription Key

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

## Step 1: Install the Example Application

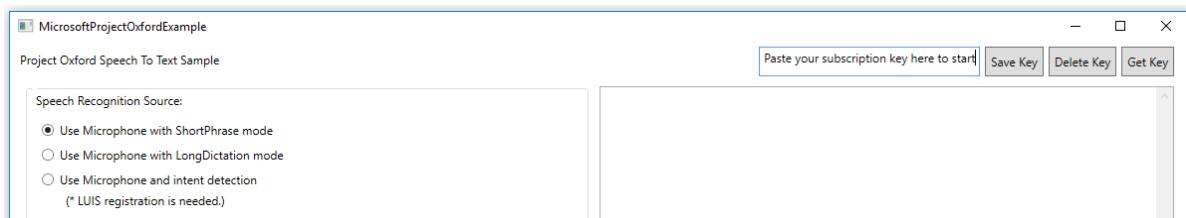
1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Browse to the folder where you saved the downloaded Speech API files. Click on **Speech**, then **Windows**, and then the **Sample-WPF** folder.
3. Double-click to open the Visual Studio 2015 Solution (.sln) file named **SpeechToText-WPF-Samples.sln**. This will open the solution in Visual Studio.

## Step 2: Build the Example Application

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

## Step 3: Run the Example Application

1. After the build is complete, press **F5** or click **Start** on the ribbon menu to run the example.
2. Locate the **Project Oxford Speech to Text** window with the **text edit box** reading "**Paste your subscription key here to start**". Paste your subscription key into the text box as shown in below screenshot. You may 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 **Speech Recognition Source** choose one of the six speech sources, which fall into two main input categories.

- Using your computer's microphone, or an attached microphone, to capture speech.
- Playing an audio file.

Each category has three recognition modes.

- **ShortPhrase Mode:** An utterance up to 15 seconds long. As data is sent to the server, the client will receive multiple partial results and one final multiple N-best choice result.
- **LongDictation Mode:** An utterance up to 2 minutes long. As data is sent to the server, the client will receive multiple partial results and multiple final results, based on where the server indicates sentence pauses.
- **Intent Detection:** The server returns additional structured information about the speech input. To use Intent you will need to first train a model. See details [here](#).

There are example audio files to be used with this example application. You find the files in the repository you downloaded with this example under **SpeechToText**, in the **Windows** folder, under **samples**, in the **SpeechRecognitionServiceExample** folder. These example audio files will run automatically if no other files are chosen when selecting the **Use wav file for Shortphrase mode** or **Use wav file for Longdictation mode** as your speech input. Currently only wav audio format is supported.

Speech Recognition Source:

- Use Microphone with ShortPhrase mode
- Use Microphone with LongDictation mode
- Use Microphone and intent detection (\* LUIS registration is needed.)
- Use wav file for ShortPhrase mode
- Use wav file for LongDictation mode
- Use wav file and intent detection (\* LUIS registration is needed.)

Start Recognition

```

--- Start speech recognition using microphone with ShortPhrase mode in en-us language ----

--- Microphone status change received by OnMicrophoneStatus() ---
***** Microphone status: True *****
Please start speaking.

--- Partial result received by OnPartialResponseReceivedHandler() ---
how

--- Partial result received by OnPartialResponseReceivedHandler() ---
hell

--- Partial result received by OnPartialResponseReceivedHandler() ---
hello

--- Partial result received by OnPartialResponseReceivedHandler() ---
hello do

--- Partial result received by OnPartialResponseReceivedHandler() ---
hello joan

--- Partial result received by OnPartialResponseReceivedHandler() ---
hello jones

--- Microphone status change received by OnMicrophoneStatus() ---
***** Microphone status: False *****

```

OnMicShortPhraseResponseReceivedHandler ---  
\*\*\*\*\* Final n-BEST Results \*\*\*\*\*  
[0] Confidence=High,Text="Hello Jones."

## Review and Learn

### Events

#### Partial Results Event:

This event gets called every time the Speech Recognition Server has an idea of what the speaker might be saying – even before he or she has finished speaking (if you are using the Microphone Client) or have finished transferring data (if you are using the Data Client).

**Intent Event:**

Called on WithIntent clients (only in ShortPhrase mode) after the final reco result has been parsed into structured JSON intent.

**Result Event:**

When you have finished speaking (in ShortPhrase mode), this event is called. You will be provided with n-best choices for the result. In LongDictation mode, the handler associated with this event will be called multiple times, based on where the server thinks sentence pauses are.

Eventhandlers are already pointed out in the code in form of code comments.

RETURN FORMAT	DESCRIPTION
<b>LexicalForm</b>	This form is optimal for use by applications that need raw, unprocessed speech recognition results.
<b>DisplayText</b>	The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user.
<b>Inverse Text Normalization (ITN) has been applied</b>	An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that display the speech recognition results to a user.
<b>InverseTextNormalizationResult</b>	Inverse text normalization (ITN) converts phrases like "one two three four" to a normalized form such as "1234". Another example is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that interpret the speech recognition results as commands or perform queries based on the recognized text.
<b>MaskedInverseTextNormalizationResult</b>	The recognized phrase with inverse text normalization and profanity masking applied, but no capitalization or punctuation. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user. Inverse Text Normalization (ITN) has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that use the unmasked ITN results but also need to display the command or query to the user.

## Related Topics

- [Get started with Bing Speech Recognition and/or intent in Java on Android](#)
- [Get started with Bing Speech Recognition and/or intent in Objective C on iOS](#)
- [Get started with Bing Speech API in JavaScript](#)
- [Get started with Bing Speech API in cURL](#)

# Get Started with Bing Speech Recognition Service Library in C# for .Net Windows

4/12/2017 • 5 min to read • [Edit Online](#)

With Microsoft Speech Recognition Service Library, your service can utilize the power of Microsoft Speech transcription cloud to convert spoken language to text. This service-to-service library works in real-time so your client app can send audio to servers in the cloud and start receiving partial recognition results back simultaneously and asynchronously. For library API reference, see the [Microsoft Bing Speech SDK](#).

## Speech Recognition Service C# Sample

This section describes how to install, build, and run the C# sample application.

### Prerequisites

- **Platform Requirements:** The below example has been developed for Windows 8+ and .NET 4.5+ Framework using [Visual Studio 2015, Community Edition](#).
- **Get the Service Library and Sample Application:** The library is available through a [Nuget Package](#). You may clone the sample through [Github](#).
- **Subscribe to Speech API and Get a Free Trial Subscription Key:** Before creating the example, you must subscribe to Speech API which is part of Microsoft Cognitive Services (previously Project Oxford). For subscription and key management details, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial.

1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Double-click to open the Visual Studio 2015 Solution (.sln) file named **SpeechClient.sln**. This will open the solution in Visual Studio.

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. Open the output directory for the sample (**SpeechClientSample\bin\Debug**), press **Shift+Right Click**, press **Open command window here**.
  3. Run **SpeechClientSample.exe** with the following arguments:
    - Arg[0]: Specify an input audio wav file.
    - Arg[1]: Specify the audio locale.
    - Arg[2]: Specify the service uri.
    - Arg[3]: Specify the subscription key to access the Speech Recognition Service.
- **ShortPhrase mode:** an utterance up to 15 seconds long. As data is sent to the server, the client will receive multiple partial results and one final best result.
  - **LongDictation mode:** an utterance up to 10 minutes long. As data is sent to the server, the client will receive multiple partial results and multiple final results, based on where the server indicates sentence pauses.

RECOGNITION MODE	SERVICE URI
Short-Form	wss://speech.platform.bing.com/api/service/recognition

RECOGNITION MODE	SERVICE URI
Long-Form	wss://speech.platform.bing.com/api/service/recognition/continuous

The Voice API supports audio/wav using the following codecs:

- PCM single channel
- Siren
- SirenSR

To create a SpeechClient, you need to first create a Preferences object. The Preferences object is a set of parameters that configures the behavior of the speech service. It consists of the following fields:

- **SpeechLanguage:** The locale of the audio being sent to the speech service.
- **ServiceUri:** The endpoint use to call the speech service.
- **AuthorizationProvider:** An IAuthorizationProvider implemetation used to fetch tokens in order to access the speech service. Although the sample provides a Cognitive Services authorization provider, it is highly recommended to create your own implementation to handle token caching.
- **EnableAudioBuffering:** An advanced option, please see [Connection Management](#)

The SpeechInput object consists of 2 fields:

- **Audio:** A stream implementation of your choice that the SDK will pull audio from. Please note that this could be any [Stream](#) that supports reading. **Note:** the SDK detects the end of the stream when it the stream returns **0** when attempting to read from it.
- **RequestMetadata:** Metadata about the speech request. For more details refer to the documentation.

Once you have instantiated a SpeechClient and SpeechInput objects, use RecognizeAsync to make a request to the speech service.

#### **var task = speechClient.RecognizeAsync(speechInput);**

The task returned by RecognizeAsync completes once the request completes. The last RecognitionResult that the server thinks is the end of the recognition. The task can Fault if the server or the SDK fails unexpectedly.

#### Partial Results Event:

This event gets called every time the Speech Recognition Server has an idea of what the speaker might be saying – even before the user has finished speaking (if you are using the Microphone Client) or have finish transferring data (if you are using the Data Client). You can subscribe to the event using

#### **SpeechClient.SubscribeToPartialResult();**

Or use the generic events subscription method

#### **SpeechClient.SubscribeTo();**

RETURN FORMAT	DESCRIPTION
<b>LexicalForm</b>	This form is optimal for use by applications that need raw, unprocessed speech recognition results.
<b>DisplayText</b>	The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user.

RETURN FORMAT	DESCRIPTION
<b>Confidence</b>	Indicates the level of confidence the recognized phrase represents the audio associated as defined by the Speech Recognition Server.
<b>MediaTime</b>	The current time relative to the start of the audio stream (In 100-nanosecond units of time).
<b>MediaDuration</b>	The current phrase duration/length relative to the audio segment (In 100-nanosecond units of time).

**Result Event:**

When you have finished speaking (in ShortPhrase mode), this event is called. You will be provided with n-best choices for the result. In LongDictation mode, the event can be called multiple times, based on where the server thinks sentence pauses are. You can subscribe to the event using

**SpeechClient.SubscribeToRecognitionResult();**

Or Use the generic events subscription method

**SpeechClient.SubscribeTo();**

RETURN FORMAT	DESCRIPTION
<b>RecognitionStatus</b>	The status on how the recognition was produced. For example, was it produced as a result of successful recognition, or as a result of canceling the connection, etc.
<b>Phrases</b>	The set of n-best recognized phrases with the recognition confidence. Refer to the above table for phrase format.

Speech Response example:

```
--- Partial result received by OnPartialResult
---what
--- Partial result received by OnPartialResult
--what's
--- Partial result received by OnPartialResult
---what's the web
--- Partial result received by OnPartialResult
---what's the weather like
----***** Phrase Recognition Status = [Success]
***What's the weather like? (Confidence:High)
What's the weather like? (Confidence:High)
```

The APIs utilizes a single web-socket connection per request. For optimal user experience, the SDK will attempt to reconnect to the speech service and start the recognition from the last RecognitionResult that it received. For example, if the audio request is 2 minutes long and the SDK received a RecognitionEvent at the 1 minute mark, then a network failure occurred after 5 seconds, the SDK will start a new connection starting from the 1 minute mark.

**NOTE**

The SDK does not seek back to the 1 minute mark, as the Stream may not support seeking. Instead the SDK keep internal buffer that it uses to buffer the audio and clears the buffer as it received RecognitionResult events.

By default, the SDK buffers audio so it can recover when a network interrupt occurs. In some scenario where it is

preferable to discard the audio lost during the network disconnect and restart the connection where the stream at due to performance considerations, it is best to disable audio buffering by setting **EnableAudioBuffering** in the Preferences object to **false**.

# Get started with Bing Speech Recognition in C# for .Net Universal Apps on Windows 10 (including Phone)

4/12/2017 • 4 min to read • [Edit Online](#)

Learn to create and develop a simple Windows 10 Universal Application that uses the [Windows.Media.SpeechRecognition API](#) to convert spoken audio input into text by sending audio to Microsoft's servers in the cloud. Alternatively you have a choice of using a REST API, which requires batching up all of your audio into a single buffer, uploading the full audio buffer, and getting a recognition result text back. Documentation for the REST API can be found [here](#). Using the [Windows.Media.SpeechRecognition API](#) allows for real-time streaming, so that as the audio is being spoken and streamed to the server, partial recognition results are returned back to the application. The rest of this tutorial describes the use of the [Windows.Media.SpeechRecognition API](#). A sample Windows 10 Universal Application project (to be used with Visual Studio 2015) illustrating basic [Windows.Media.SpeechRecognition API](#) usage can be found for your reference. Download [Universal APP SDK](#).

## Prerequisites

1. Install Windows 10 from [here](#).
2. Install Visual Studio 2015 from [here](#). During the installation of Visual Studio 2015, make sure that you select the **Custom installation** option and check all the **Universal Windows App Development Tools** options.
3. Upgrade your Visual Studio 2015 to [Update 2](#).

If you have a **pre-existing installation of Visual Studio 2015** and are unsure if the **Universal Windows App Development Tools** options were set, do the following.

1. Go to **Control Panel > Programs > Programs and Features** and look for your Microsoft Visual Studio 2015 entry.
2. Right-click on the **Visual Studio 2015** application and choose **Change**.
3. Wait a bit and, in the upcoming Visual Studio 2015 dialog box, choose **Modify**. Look in the upcoming list of options for **Universal Windows App Development Tools** and click on that to reveal all the options under **Universal Windows App Development Tools** to select all of them.
4. Hit **Next** and then **Update** to install [Update 2](#).

## Step 1: Create an Windows 10 Universal Application Project

In this step you will create a Windows 10 Universal application project where you will use the [Windows.Media.SpeechRecognition API](#)

1. Open Visual Studio 2015.
2. From the **File** menu, click **New** and then **Project**. In the **New Project** dialog box, click **Visual C# > Windows > Windows Universal > Blank App (Windows Universal)**.

## Step 2: Add Speech Recognition API Use in Your Application

Now, you will add use of the Bing Speech Recognition API in your application to convert spoken audio input into text. You can look at the code in MainPage.xaml.cs [Universal APP SDK](#) for reference.

1. Create a `SpeechRecognizer` object.
2. Create at least one `SpeechRecognitionConstraint` type object and add to the `SpeechRecognizer` object created above. For our simple dictation scenario, we use the pre-defined `SpeechRecognitionTopicConstraint`

constraint type.

### 3. Compile the constraints.

```
m_recognizer = new SpeechRecognizer();
SpeechRecognitionTopicConstraint topicConstraint = new
SpeechRecognitionTopicConstraint(SpeechRecognitionScenario.Dictation, "Development");
m_recognizer.Constraints.Add(topicConstraint);
SpeechRecognitionCompilationResult result = await m_recognizer.CompileConstraintsAsync();
```

SpeechRecognizer supports 2 types of recognition sessions:

- **Continuous recognition sessions** for prolonged audio input from the user. Here, recognition results for phrases spoken during the session are continually produced and returned to the user while the user continues speaking. A continuous session needs to be either explicitly ended or automatically times out after a configurable period of silence (default is 20 seconds).
- **Speech recognition session for recognizing** a short phrase. The session is terminated and the recognition results returned when a pause is detected by the recognizer.

The objects corresponding to each of these types of sessions have various events for which handlers can be attached to obtain notification of various interesting events during the recognition session.

A continuous recognition session can be started by calling the `SpeechRecognizer.ContinuousRecognitionSession.StartAsync` method. The `SpeechRecognizer.ContinuousRecognitionSession` object provides the following events:

- **ResultGenerated event:** Triggered every time results of a phrase spoken during the session are available. The results are passed as an argument to any handlers attached to the event.
- **Completed event:** Triggered at the end of a continuous recognition session.

```
m_recognizer.ContinuousRecognitionSession.Completed += 
OnContinuousRecognitionSessionCompletedHandler;

m_recognizer.ContinuousRecognitionSession.ResultGenerated += 
OnContinuousRecognitionSessionResultGeneratedHandler;

m_recognizer.ContinuousRecognitionSession.StartAsync()
```

```
var recognition = m_recognizer.RecognizeAsync();
recognition.Completed += this.OnRecognitionCompletedHandler;
```

A speech recognition session for short phrase recognition can be started by calling the `SpeechRecognizer.RecognizeAsync` method. This returns an `IAsyncOperation<SpeechRecognitionResult>` object, which provides the `Completed` event that is triggered upon completion of the recognition session. The results are passed as an argument to any handlers attached to the `Completed` event.

For the short phrase and continuous recognition scenarios, results are available in a `SpeechRecognitionResult` object accessible through the arguments of the `Completed` and `ResultGenerated` event handlers respectively. This object provides n-best alternatives in decreasing order of quality (results with highest recognition confidence level is first followed by results with decreasing recognition confidence levels).

Additionally, the `SpeechRecognizer` object itself includes a `HypothesisGenerated` event that is triggered every time partial results (or hypothesis) of the recognition is available before the final recognition results are produced. This event applies to both short phrase and continuous speech recognition sessions. Partial recognition results are passed as an argument to any handlers attached to the event.

## Step 3: Deploying and Running Your Application

This part describes some of the necessary steps for successfully running the speech recognition feature in your app.

1. Add the **Microphone** capability in your project's Package.appxmanifest (see the example project for reference)
2. If you run the app on Windows 10 Mobile, enable the speech recognition service on your Windows Phone device or the emulator by going to **Settings > Privacy > Speech, inking & typing**, and enabling the **Get to know me** feature.

On desktop and other Windows 10 universal devices, you will receive a prompt on starting the application, asking permission to use the speech recognition service.

In conclusion, we should note that this tutorial and the example project provided here only illustrates basic functionality of the [Windows.Media.SpeechRecognition API](#). These APIs offer a much richer set of features that we encourage you to explore through the API documentation on MSDN.

## Related Topics

- [Get Started with Speech Recognition in C Sharp for .Net on Windows Desktop](#)
- [Get Started with Speech Recognition in C Sharp for .Net on Windows Phone 8.1](#)
- [Get started with Speech Recognition in Java on Android](#)
- [Get started with Speech Recognition in Objective C on iOS](#)
- [Get started with Bing Speech API in cURL](#)

# Get Started with Bing Speech Recognition in C# for .Net on Windows Phone 8.1

4/12/2017 • 3 min to read • [Edit Online](#)

Develop a basic Windows Phone 8.1 application that uses the [Windows.Media.SpeechRecognition API Client Library](#) to convert spoken audio input into text by sending audio to Microsoft's servers in the cloud.

Using the Windows.Media.SpeechRecognition API Client Library allows for real-time streaming, which means that at the same time your client application sends audio to the service, it simultaneously and asynchronously receives partial recognition results back. This tutorial describes the use of the [Windows.Media.SpeechRecognition API Client Library](#).

## Platform requirements

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

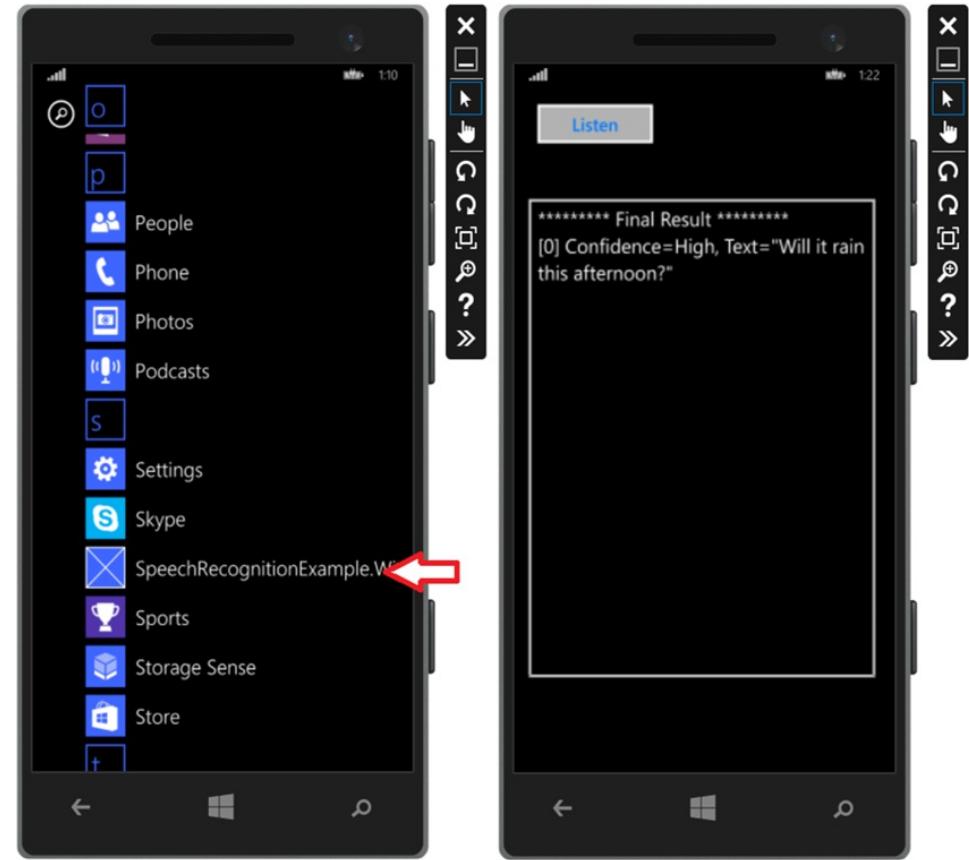
## Access the client library and download the example

You can access the Windows Media Speech Recognition Client Library through this [link](#) and you may download the example application for a Windows Phone 8.1 Universal project [here](#). The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder. You may also need to download the Windows Phone tools, which is an optional add-on to Visual Studio, if it is not already installed.

## Subscribe to Speech API and get a free trial subscription key

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

1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
  2. Browse to the folder where you saved the downloaded **SpeechRecognitionExample.WindowsPhone8.1** files. Click to open the **SpeechRecognitionExample.WindowsPhone8.1** folder.
  3. Double-click to open the Visual Studio 2015 Solution (.sln) file named **SpeechRecognitionExample.WindowsPhone8.1.sln**. This will open the solution in Visual Studio.
- 
1. Press Ctrl+Shift+B, or click **Build** on the ribbon menu, then select **Build Solution**.
  2. Note, you may have to specifically add the **SpeechRecognitionExample.WindowsPhone8.1.csproj** file to the solution, if it does not run automatically. The .csproj file can be found in the same folder as **SpeechRecognitionExample.WindowsPhone8.1.sln**.
- 
1. Before running the example application, you need to decide whether to use a device or a phone emulator for running your example application. If you choose the emulator, make sure your system allows virtualization\* and has virtualization software such as Hyper-V installed to simulate the phone hardware. (\*Virtualization is turned off by default in Windows 8, Windows 8.1, and Windows 10.)
  2. Click **Start** on the ribbon menu to select **Device** or one of the **Emulators** to run the example.
  3. Locate the **Windows Phone** window or your chosen device, and scroll the apps. You should find the **Speech Recognition Example** under "S".



1. Tap the app to open it. A simple user interface opens. Tap the "Listen" button and speak a sentence or two. (Make sure the microphone is on.) The spoken audio should be returned as text and appear in the square window immediately after your speech. A confidence level accompanies the returned text.

## Review and Learn

One SpeechRecognizer object can be used for multiple recognition sessions. (BH comment: Suggest adding the preceding line of code which contains the async modifier: )

```
protected async override void OnNavigatedTo(NavigationEventArgs e)
![WindowsPhone code](./Images/WindowsPhone-codeSample.PNG)
```

```
m_recognizer = new SpeechRecognizer();
SpeechRecognitionTopicConstraint topicConstraint = new
SpeechRecognitionTopicConstraint(SpeechRecognitionScenario.Dictation, "Development");
m_recognizer.Constraints.Add(topicConstraint);
SpeechRecognitionCompilationResult result = await m_recognizer.CompileConstraintsAsync();
```

```
var recognition = m_recognizer.RecognizeAsync();
recognition.Completed += this.OnRecognitionCompletedHandler;
```

A speech recognition session can be started by calling the SpeechRecognizer.RecognizeAsync method. This returns an IAsyncOperation< SpeechRecognitionResult > object, which provides the Completed event that is triggered upon completion of the recognition session. The session is terminated and the recognition results returned when a pause is detected by the recognizer. The results are passed as an argument to any handlers attached to the Completed event.

The results are available in a `SpeechRecognitionResult` object accessible through the arguments of the `Completed` event handler. This object provides n-best alternatives in decreasing order of quality (results with highest recognition confidence level is first followed by results with decreasing recognition confidence levels).

## Summary Remarks

In conclusion, this "Get Started" introduction and the provided example application only illustrates basic functionality of the Windows Media Speech Recognition API. The API offers a rich set of features that we encourage you to explore through the API documentation on MSDN and further experimentation.

## Related Topics

- [Get Started with Bing Speech Recognition in C Sharp for .Net on Windows Desktop](#)
- [Get Started with Bing Speech Recognition in Java on Android](#)
- [Get Started with Bing Speech Recognition in Objective C on iOS](#)
- [Get Started with Bing Speech API in JavaScript](#)
- [Get Started with Bing Speech API in cURL](#)

# Get Started with Bing Speech Recognition in Java on Android

4/12/2017 • 6 min to read • [Edit Online](#)

With Bing Speech Recognition API you can develop Android applications that leverage Microsoft cloud servers to convert spoken audio to text. The API supports real-time streaming, so your application can simultaneously and asynchronously receive partial recognition results at the same time it is sending audio to the service.

This article uses a sample application to demonstrate how to use the Bing Speech Recognition API Client Library for Android to develop speech to text applications in Java for Android devices.

## Prerequisites

### Platform Requirements

The below example has been developed for [Android Studio](#) for Windows in Java.

### Get the Client Library and Example Application

Download Speech Recognition API Client Library for Android from [this link](#). The downloaded files need to be saved to a folder of your choice. Inside there is both a fully buildable example and the SDK library. The buildable example can be found under **samples** in the **SpeechRecoExample** directory. The two libraries you need to use in your own apps can be found in the **SpeechSDK** folder under **libs** in the **armeabi** and the **x86** folder. The size of **libandroid\_platform.so** file is 22 MB but gets reduced to 4MB at deploy time.

### Subscribe to Speech API and Get a Free Trial Subscription Key

Before creating the example, you must subscribe to Speech API which is part of Cognitive Services. Click the yellow "**Try for free**" button on one of the offered services, in this case Speech API, and follow the directions. For subscription and key management details, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial.

## Step 1: Install the Example Application and Create the Application Framework

Create an Android application project to implement use of the Speech Recognition API

1. Open Android Studio.
2. Paste your subscription key into the **primaryKey** string in the `..\samples\SpeechRecoExample\res\values` folder.

### NOTE

You don't have to worry about the LUIS values if you don't want to use Intent at this point.)

3. Create a new application project.
4. Using files downloaded from the **speech\_SpeechToText-SDK-Android** zip package, do the following:
  - a. Copy the **speechsdk.jar** file, found in the **SpeechSDK** folder inside the **Bin** folder, to the "**your-application\app\libs**" folder.
  - b. Right click "**app**" in the project tree, select "**Open module settings**", select the "**Dependencies**" tab, and click "+" to add a "**File dependency**".
  - c. Select the **libs\speechsdk.jar** in the "**Select Path**" dialog box.

- d. Copy the **libandroid\_platform.so** file to the "**your-application\app\src\main\jniLibs\armeabi**" folder.

**You can now run the example application or continue with thes following instructions to build your own application.**

## Step 2: Build the Example Application

Open [MainActivity.java](#) or locate the **MainActivity.java** file within the **samples, SpeechRecoExample, src, com, microsoft, AzureIntelligentServicesExample** folder from the downloaded **speech\_SpeechToText-SDK-Android** zip package. You will need the subscription key you generated above. Once you have added your subscription key to the application, notice that you use the **SpeechRecognitionServiceFactory** to create a client of your liking.

```

void initializeRecoClient()
{
    String language = "en-us";

    String subscriptionKey = this.getString(R.string.subscription_key);
    String luisAppID = this.getString(R.string.luisAppID);
    String luisSubscriptionID = this.getString(R.string.luisSubscriptionID);

    if (_isMicrophoneReco && null == _micClient) {
        if (!_isIntent) {
            _micClient = SpeechRecognitionServiceFactory.createMicrophoneClient(this,
                _recoMode,
                language,
                this,
                subscriptionKey);
        }
        else {
            MicrophoneRecognitionClientWithIntent intentMicClient;
            intentMicClient = SpeechRecognitionServiceFactory.createMicrophoneClientWithIntent(this,
                language,
                this,
                subscriptionKey,
                luisAppID,
                luisSubscriptionID);
            _micClient = intentMicClient;
        }
    }
    else if (!_isMicrophoneReco && null == _dataClient) {
        if (!_isIntent) {
            _dataClient = SpeechRecognitionServiceFactory.createDataClient(this,
                _recoMode,
                language,
                this,
                subscriptionKey);
        }
        else {
            DataRecognitionClientWithIntent intentDataClient;
            intentDataClient = SpeechRecognitionServiceFactory.createDataClientWithIntent(this,
                language,
                this,
                subscriptionKey,
                luisAppID,
                luisSubscriptionID);
            _dataClient = intentDataClient;
        }
    }
}

```

#### Create a Client:

Create one of the following clients

- **DataRecognitionClient:** Speech recognition with PCM data, for example from a file or audio source. The data is broken up into buffers and each buffer is sent to the Speech Recognition Service. No modification is done to the buffers, so the user can apply their own Silence Detection if desired. If the data is provided from wave files, you can send data from the file directly to the server. If you have raw data, for example audio coming over Bluetooth, then you first send a format header to the server followed by the data.
- **MicrophoneRecognitionClient:** Speech recognition with audio coming from the microphone. Make sure the microphone is turned on and data from the microphone is sent to the Speech Recognition Service. A built-in Silence Detector is applied to the microphone data before it is sent to the recognition service.

- **"WithIntent" clients:** Use "WithIntent" if you want the server to return additional structured information about the speech to be used by apps to parse the intent of the speaker and drive further actions by the app. To use Intent, you will need to train a model and get an AppID and a Secret. See project [LUIS](#) for details.

#### Select a Locale

When you use the SpeechRecognitionServiceFactory to create the Client, you must select a language.

Supported locales include:

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
de-DE	zh-TW	zh-HK	ru-RU
es-ES	ja-JP	ar-EG*	da-DK
en-GB	en-IN	fi-FI	nl-NL
en-US	pt-BR	pt-PT	ca-ES
fr-FR	ko-KR	en-NZ	nb-NO
it-IT	fr-CA	pl-PL	es-MX
zh-CN	en-AU	en-CA	sv-SE

\*ar-EG supports Modern Standard Arabic (MSA)

#### Select a Recognition Mode

You also need to provide the recognition mode.

- **ShortPhrase mode:** An utterance up to 15 seconds long. As data is sent to the service, the client will receive multiple partial results and one final multiple n-best choice result.
- **LongDictation mode:** An utterance up to 2 minutes long. As data is sent to the service, the client will receive multiple partial results and multiple final results, based on where the server identifies sentence pauses.

#### Attach an Event Handler

From the created client, you can attach various event handlers.

- **Partial Results Events:** This event gets called every time the Speech Recognition Server has an idea of what you might be saying – even before you finish speaking (if you are using the Microphone Client) or have finished sending up data (if you are using the Data Client).
- **Error Events:** Called when the Server Detects Error
- **Intent Events:** Called on WithIntent clients (only in ShortPhrase mode) after the final reco result has been parsed into a structured JSON intent.
- **Result Events:** When you have finished speaking (in ShortPhrase mode), this event is called. You will be provided with n-best choices for the result. In LongDictation mode, the handler's associated with this event will be called multiple times, based on where the server thinks sentence pauses are.

#### Select Confidence Value and Text Form

For each of the n-best choices, you get a confidence value and few different forms of the recognized text:

- **LexicalForm:** This form is optimal for use by applications that need the raw, unprocessed speech recognition result.
- **DisplayText:** The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, e.g. "d\*\*\*". This form is optimal for use by applications that display the speech recognition results to a user.

- **Inverse Text Normalization (ITN):** has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that display the speech recognition results to a user.
- **InverseTextNormalizationResult:** Inverse text normalization (ITN) converts phrases like "one two three four" to a normalized form such as "1234". Another example is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that interpret the speech recognition results as commands or which perform queries based on the recognized text.
- **MaskedInverseTextNormalizationResult:** The recognized phrase with inverse text normalization and profanity masking applied, but not capitalization or punctuation. Profanity is masked with asterisks after the initial character, e.g. "d\*\*\*". This form is optimal for use by applications that display the speech recognition results to a user. Inverse Text Normalization (ITN) has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that use the unmasked ITN results but also need to display the command or query to the user.

## Step 3: Run the Example Application

Run the application with the chosen clients, recognition modes and event handlers.

## Related Topics

- [Get Started with Bing Speech Recognition in C Sharp for Windows in .NET](#)
- [Get Started with Bing Speech Recognition and/or intent in Objective C on iOS](#)
- [Get Started with Bing Speech API in JavaScript](#)

# Get Started with Bing Speech API in JavaScript

4/12/2017 • 2 min to read • [Edit Online](#)

With Bing Speech API you can develop basic JavaScript applications that leverage Microsoft's cloud servers to convert spoken audio to text. This article describes an example web application created in Visual Studio 2015 that demonstrates the use of Bing Speech API using Javascript.

The Speech Recognition example demonstrates the following features using a wav file or external microphone input:

- Short-form recognition
- Recognition with intent

To use SpeechJS, simply host Speech.1.0.0.js on your website. A 'minified' version of SpeechJS is also available Speech.1.0.0.min.js.

## Prerequisites

### Platform requirements

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

### Get the client library and example

You may download the Speech API client library and example through [GitHub](#). The downloaded folder needs to be hosted locally on your machine to follow the below scenario.

### Subscribe to Speech API and get a free trial subscription key

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

## Step 1: Install the Example Application

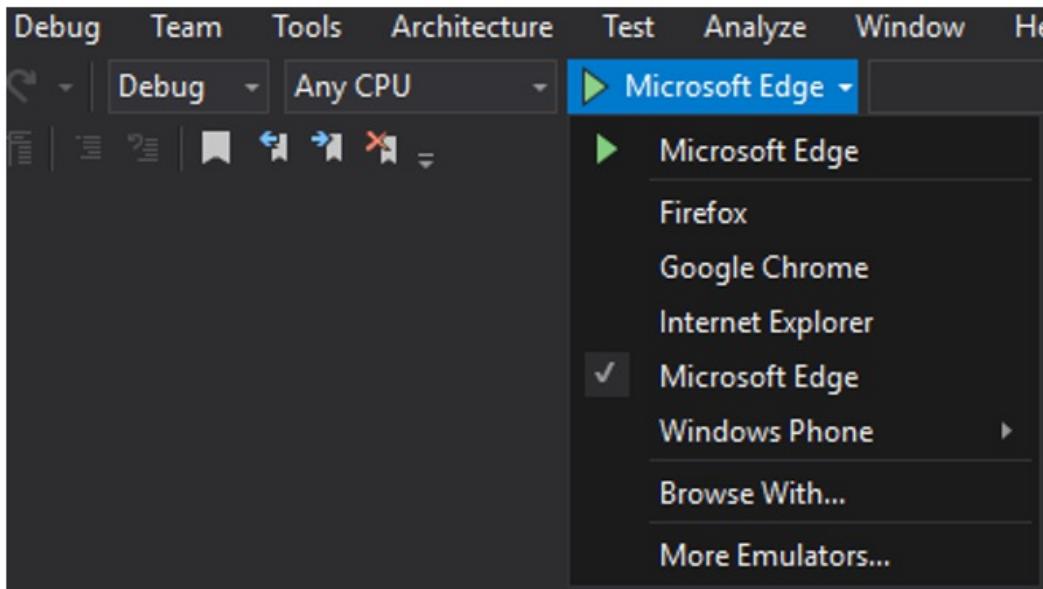
1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Browse to the folder where you saved the downloaded SpeechJS files. Click on **Speech** and then the **Speech.JS** folder.
3. Double-click to open the Visual Studio 2015 Solution (.sln) file named **Oxford.SpeechJS.sln**. This will open the solution in Visual Studio.

## Step 2: Build the Example Application

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

## Step 3: Run the Example Application

1. After the build is complete, choose the target browser you would like to use for running your Speech-to-Text web app.



2. Press **F5** or click **Start** on the ribbon menu to run the example.
3. Locate the **Cognitive Services Speech to Text** window with the **text edit box** reading "**Subscription**". Paste your subscription key into the subscription text box.
4. Check if you would like to use the Microphone as your voice input and what speech mode you would like to use by selecting a setting in the "Mode" drop-down box.
5. For modes where you would like both Speech Recognition and Intent to work together, you need to first sign up for [Language Understanding Intelligent Service \(LUIS\)](#), then set the key values in the fields "LUIS AppID" and "LUIS SubscriptionId".

## Related Topics

- [Get Started with Bing Speech Recognition in C Sharp for .Net on Windows Desktop](#)
- [Get started with Bing Speech Recognition and/or intent in Java on Android](#)
- [Get started with Bing Speech Recognition and/or intent in Objective C on iOS](#)
- [Get started with Bing Speech API in cURL](#)

For questions, feedback, or suggestions about Microsoft Cognitive Services, feel free to reach out to us directly.

- Cognitive Services [UserVoice Forum](#)

### License

All Microsoft Cognitive Services SDKs and samples are licensed with the MIT License. For more details, see [LICENSE](#).

# Bing Speech Recognition API

4/12/2017 • 8 min to read • [Edit Online](#)

This documentation describes the Bing Speech Recognition REST API that exposes an HTTP interface which enables developers to transcribe voice queries. The Bing Speech Recognition API may be used in many different contexts that need cloud-based speech recognition capabilities.

Every request requires a JSON Web Token (JWT) access token. The JWT access token is passed through in the Speech request header. The token has an expiry time of 10 minutes. See [Get Started for Free](#) for information about subscribing and obtaining API keys used to retrieve valid JWT access tokens.

The API key is passed to the token service, for example:

```
POST https://api.cognitive.microsoft.com/sts/v1.0/issueToken
Content-Length: 0
```

The required header for token access is:

NAME	FORMAT	DESCRIPTION, EXAMPLE AND USE
Ocp-Apim-Subscription-Key	ASCII	Your subscription key.

The token service returns the JWT access token as text/plain. Then the JWT is passed as a Base64 access\_token to the Speech endpoint as an Authorization header prefixed with the string Bearer, for example:

```
Authorization: Bearer [Base64 access_token]
```

Clients must use the following endpoint to access the service and build voice enabled applications:

```
https://speech.platform.bing.com/recognize
```

## NOTE

Until you have acquired an [access token](#) with your subscription key as described above this link will generate a 403 Response Error.

The API uses HTTP POST to upload audio. The API supports [chunked transfer encoding](#) for efficient audio streaming. For live transcription scenarios, it is recommended you use chunked transfer encoding to stream the audio to the service while the user is speaking. Other implementations result in higher user-perceived latency.

Your application must indicate the end of the audio to determine start and end of speech, which in turn is used by the service to determine the start and end of the request. You may not upload more than 10 seconds of audio in any one request and the total request duration cannot exceed 14 seconds.

Inputs to the Bing Speech Recognition API are expressed as HTTP query parameters. Parameters in the POST body are treated as audio content. Unsafe characters should be escaped following the W3C URL spec (<http://www.w3.org/Addressing/URL/url-spec.txt>). A request with more than one instance of any parameter will result in an error response (HTTP 400).

Following is a complete list of required and optional input parameters.



NAME	FORMAT	DESCRIPTION, EXAMPLE AND USE
Version	Digits and ":"	The API version being used by the client. The required value is 3.0. <b>Example:</b> version=3.0
requestid	GUID	A globally unique identifier generated by the client for this request. <b>Example:</b> requestid=b2c95ede-97eb-4c88-81e4-80f32d6aee54
appId	GUID	A globally unique identifier used for this application. Always use appId = D4D52672-91D7-4C74-8AD8-42B1D98141A5. Do not generate a new GUID as it will be unsupported.
format	UTF-8	Specifies the desired format of the returned data. The required value is JSON. <b>Example:</b> format=json.
locale	UTF-8 (IETF Language Tag)	Language code of the audio content in IETF RFC 5646. Case does not matter. <b>Example:</b> locale=en-US.
device.os	UTF-8	Operating system the client is running on. This is an open field but we encourage clients to use it consistently across devices and applications. Suggested values: Windows OS, Windows Phone OS, XBOX, Android, iPhone OS. <b>Example:</b> device.os=Windows Phone OS.
scenarios	UTF-8	The context for performing a recognition. The supported values are: ulm, websearch. <b>Example:</b> scenarios=ulm.
instanceid	GUID	A globally unique device identifier of the device making the request. <b>Example:</b> instanceid=b2c95ede-97eb-4c88-81e4-80f32d6aee54

#### NOTE

The values below are used either for performing the request or to manage the service operationally.

NAME	FORMAT	DESCRIPTION AND EXAMPLE
maxnbest	Integer	Maximum number of results the voice application API should return. The default is 1. The maximum is 5. <b>Example:</b> maxnbest=3

NAME	FORMAT	DESCRIPTION AND EXAMPLE
result.profanitymarkup	0/1	Scan the result text for words included in an offensive word list. If found, the word will be delimited by bad word tag. <b>Example:</b> result.profanity=1 (0 means off, 1 means on, default is 1.)

A working code sample of REST API implementation can be found [here](#).

The following is an example of a request where the audio is supplied as part of a recognition request:

```
POST /recognize?scenarios=catsearch&appid=f84e364c-ec34-4773-a783-73707bd9a585&locale=en-US&device.os=wp7&version=3.0&format=xml&requestid=1d4b6030-9099-11e0-91e4-0800200c9a66&instanceid=1d4b6030-9099-11e0-91e4-0800200c9a66 HTTP/1.1
Host: speech.platform.bing.com
Content-Type: audio/wav; samplerate=16000
Authorization: Bearer [Base64 access_token]

(audio data)
```

The Speech Recognition API supports audio/wav using the following codecs:

- PCM single channel
- Siren
- SirenSR

The API response is returned in JSON format. The value of the "name" tag has the post-inverse text normalization result. The value of the "lexical" tag has the pre-inverse text normalization result.

Schema Legend:

- < ... > means optional
- "[ ]" represents a json array
- "{ }" represents a json object
- "\*" indicates that the object can be defined multiple times

```

JSON-text: version header <results> // result must be always and only present when status = "success"
version: string // The API version "3.0" – the value the client passed and consequently the API version that serviced the request.
header: {status scenario <name> <lexical> properties} // name and lexical are always and only present in the case of "status" = "success"
    status: "success"/ "error"/ "false reco"// 'false reco' is returned only for 2.0 responses when NOSPEECH OR FALSERECHO is 1. This is done to maintain backward compatibility.
    scenario: string // the scenario this recognition result came from.
    name: string // formatted recognition result. Profane terms are surrounded with <profanity> tags.
    lexical: string // text of what was spoken. Profane terms are surrounded with <profanity> tags.
    properties: {requestid <NOSPEECH> <FALSERECHO> <HIGHCONF> <ERROR>}
        requestid: string // this is a uuid identifying the requestid to be sent with the associated logs.
It should be used as the "server.requestid" parameter value in the subsequent logging API request.
        NOSPEECH: 1 // set when no speech was detected on the sent audio.
        FALSERECHO: 1 // set when no matches were found for the sent audio.
        HIGHCONF: 1 // set when the header result is determined to be of high-confidence.
        MIDCONF: 1 // set when the header result is determined to be of medium-confidence.
        LOWCONF: 1 // set when the header result is determined to be of low-confidence.
        ERROR: 1 // set when there was an error generating a response.
results: [{scenario name lexical confidence pronunciation tokens}*] // n-best list of results ordered by confidence.
    scenario: string // the scenario this recognition result came from.
    name: string // formatted recognition result. Profane terms are surrounded with <profanity> tags.
    lexical: string // text of what was spoken. Profane terms are surrounded with <profanity> tags.
    properties: {<HIGHCONF>}
        HIGHCONF: 1 // set when the header result is determined to be of high-confidence.
        MIDCONF: 1 // set when the header result is determined to be of medium-confidence.
        LOWCONF: 1 // set when the header result is determined to be of low-confidence.

```

- **speechbox-root:**

- **child tags:** version, header, results
- **value:** none
- **attributes:** none

- **version:**

- **child tags:** none
- **value:** string, the API version "3.0"
- **attributes:** none

- **header:**

- **tags:** status, name, lexical, properties
- **value:** none
- **attributes:** none

- **status:**

- **child tags:** none
- **value:** string, "success" or "error"
- **attributes:** none

- **scenario:**

- **child tags:** none
- **value:** string, the scenario parameter value
- **attributes:** none

- **name:**

- **child tags:** none
- **value:** string, formatted recognition result

- **attributes:** none
- **lexical:**
  - **child tags:** none
  - **value:** string, text of what was spoken
  - **attributes:** none
- **properties:**
  - **child tags:** property
  - **value:** none
  - **attributes:** none
- **property:**
  - **child tags:** none
  - **value:** string, the property value
  - **attributes:** { name: the property name }
  - **other information:** Properties based in the header have the following valid names: requestid, NOSPEECH, FALSERECO. Properties based in the result portion consist of only HIGHCONF.
  - **valid property values:** requestid: a valid GUID string NOSPEECH: "1" to indicate a no speech false recognition FALSERECO: "1" to indicate that there were no interpretations HIGHCONF: "1" to indicate high confidence (currently above .5 threshold) MIDCONF: "1" to indicate medium-confidence LOWCONF: "1" to indicate low-confidence
- **results:**
  - **child tags:** result
  - **value:** none
  - **attributes:** none
- **result:**
  - **child tags:** name, lexical, confidence, tokens
  - **value:** none
  - **attributes:** none
- **confidence:**
  - **child tags:** none
  - **value:** float, indicates result confidence
  - **attributes:** none
- **tokens:**
  - **child tags:** token
  - **value:** none
  - **attributes:** none
- **token:**
  - **child tags:** name, lexical, pronunciation
  - **value:** none
  - **attributes:** none
- **pronunciation:**
  - **child tags:** none
  - **value:** string, the IPA pronunciation of this token
  - **attributes:** none

Example JSON response for a successful voice search. The comments and formatting of the JSON below is for example reasons only. The real result will omit indentations, spaces, smart quotes, comments, etc.

```
HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/json; charset=UTF-8
{

    "version": "3.0",
    "header": {
        "status": "success",
        "scenario": "websearch",
        "name": "Mc Derman Autos",
        "lexical": "mac derman autos",
        "properties": {
            "requestid": "ABDD97E-171F-4B75-A491-A977027B0BC3"
        },
        "results": [
            # Formatted result
            {
                "name": "Mc Derman Autos",
                # The text of what was spoken
                "lexical": "mac derman autos",
                # Words that make up the result; a word can include a space if there
                # isn't supposed to be a pause between words when speaking them
                "tokens": [
                    {
                        # The text in the grammar that matched what was spoken for this token
                        "token": "mc derman",
                        # The text of what was spoken for this token
                        "lexical": "mac derman",
                        # The IPA pronunciation of this token (I made up M AC DIR MANT;
                        # refer to a real IPA spec for the text of an actual pronunciation)
                        "pronunciation": "M AC DIR MANT",
                    },
                    {
                        "token": "autos",
                        "lexical": "autos",
                        "pronunciation": "OW TOS",
                    }
                ],
                "properties": {
                    "HIGHCONF": "1"
                }
            }
        ]
    }
}
```

```
</speechbox-root>
```

Example JSON response for a voice search query where the user's speech could not be recognized.

```

HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: application/json; charset=UTF-8

{
  "version": "3.0",
  "results": [{}],
  "header": {
    "status": "error",
    "scenario": "websearch",
    "properties": {
      "requestid": "ABDD97E-171F-4B75-A491-A977027B0BC3",
      "FALSERECO": "1"
    }
  }
}

```

- **Http/400 BadRequest:** Will be returned if a required parameter is missing, empty or null, or if the value passed to either a required or optional parameter is invalid. The “Invalid” response includes passing a string value that is longer than the allowed length. A brief description of the problematic parameter will be included.
- **Http/401 Unauthorized:** Will be returned if the request is not authorized.
- **Http/502 BadGateway:** Will be returned when the service was unable to perform the recognition.
- **Http/403 Forbidden:** Will be returned when there are issues with your authentication or quota.

Example response for a voice search request submitted with a bad parameter. This is the error response format regardless of what format the user has requested.

```

HTTP/1.0 400 Bad Request
Content-Length: XXX
Content-Type: text/plain; charset=UTF-8

Invalid lat parameter specified

```

Post all questions and issues to the [Bing Speech Service](#) MSDN Forum, with complete detail, such as:

- An example of the full request string (minus the raw audio data).
- If applicable, the full output of a failed request, which includes log IDs.
- The percentage of requests that are failing.

# Bing Text To Speech API

4/12/2017 • 6 min to read • [Edit Online](#)

With the Bing Text to Speech API your application can send HTTP requests to a cloud server, where text is instantly synthesized into human sounding speech, and returned as an audio file. The API can be used in many different contexts to provide real-time text to speech conversion in a variety of different voices and languages.

Every request requires a JSON Web Token (JWT) access token. The JWT access token is passed through in the Speech request header. The token has an expiry time of 10 minutes. See [Get Started for Free](#) for information about subscribing and obtaining API keys used to retrieve valid JWT access tokens.

The API key is passed to the token service, for example:

```
POST https://api.cognitive.microsoft.com/sts/v1.0/issueToken
Content-Length: 0
```

The required header for token access is:

NAME	FORMAT	DESCRIPTION, EXAMPLE AND USE
Ocp-Apim-Subscription-Key	ASCII	Your subscription key.

The token service returns the JWT access token as text/plain. Then the JWT is passed as a Base64 access\_token to the Speech endpoint as an Authorization header prefixed with the string Bearer, for example:

```
Authorization: Bearer [Base64 access_token]
```

Clients must use the following endpoint to access the text to speech service:

```
https://speech.platform.bing.com/synthesize
```

## NOTE

Until you have acquired an access token with your subscription key as described above this link will generate a 403 Response Error.

HTTP headers for the request include:

HEADER	VALUE	COMMENTS
Content-Type	application/ssml+xml	The input content type
X-Microsoft-OutputFormat	<b>1)</b> ssml-16khz-16bit-mono-tts, <b>2)</b> raw-16khz-16bit-mono-pcm, <b>3)</b> audio-16khz-16kbps-mono-siren, <b>4)</b> riff-16khz-16kbps-mono-siren <b>5)</b> riff-16khz-16bit-mono-pcm <b>6)</b> audio-16khz-128kbitrate-mono-mp3, <b>7)</b> audio-16khz-64kbitrate-mono-mp3, <b>8)</b> audio-16khz-32kbitrate-mono-mp3	The output audio format

HEADER	VALUE	COMMENTS
X-Search-AppId	A GUID (hex only, no dashes)	An ID that uniquely identifies the client application. This can be Store ID for Apps. If one is not available, the ID may be user generated per-application.
X-Search-ClientID	A GUID (hex only, no dashes)	An ID that uniquely identifies application instance per-installation.
User-Agent	Application name	Application name is required and must be less than 255 characters.

Inputs to the Bing Text to Speech API are expressed as HTTP query parameters. Parameters in the POST body are treated as Speech Synthesis Markup Language (SSML) content. Refer to the [SSML W3C Specification](#) for a description of the markup used to control aspects of speech such as the language and gender of the speaker. The following is a complete list of recognized input parameters:

PARAMETER	DESCRIPTION	VALUES
VoiceType	Indicates the preferred gender of the voice to speak the synthesized text.	Female, Male
VoiceName	Indicates the processor-specific voice name used to speak the synthesized text.	See <a href="#">Supported Locales and Voice Fonts</a>
Locale	Indicates the language used to speak the synthesized text.	See <a href="#">Supported Locales and Voice Fonts</a>
OutputFormat	Indicates the audio format the text will be synthesized into	SSML16Khz16BitMonoTTS, Raw16khz16bitMonoPCM, Audio16khz16kbpsMonoSiren, Riff16khz16kbpsMonoSiren, Riff16khz16bitMonoPcm, Audio16khz128kbitrateMonoMp3, Audio16khz64kbitrateMonoMp3, Audio16khz32kbitrateMonoMp3
RequestUri	Indicates the URI of the Internet resource associated with the request	
AuthorizationToken	Token used to validate the transaction	
Text	The text to be synthesized.	<b>Note:</b> Unsafe characters should be escaped following the W3C URL specifications ( <a href="http://www.w3.org/Addressing/URL/url-spec.txt">http://www.w3.org/Addressing/URL/url-spec.txt</a> ).

#### NOTE

Requests with more than one instance of any parameter will result in an HTTP 400 error response.

The following is an example of a voice output request:

```

POST /synthesize
HTTP/1.1
Host: speech.platform.bing.com
Content-Type: audio/wav; samplerate=8000

X-Microsoft-OutputFormat: riff-8khz-8bit-mono-mulaw
Content-Type: text/plain; charset=utf-8
Host: speech.platform.bing.com
Content-Length: 197

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Female' name='Microsoft Server
Speech Text to Speech Voice (en-US, ZiraRUS)'>Microsoft Bing Voice Output API</voice></speak>

```

The Bing Text to Speech API uses HTTP POST to send audio back to the client. The API response contains the audio stream and the codec and will match the requested output format. The maximum amount of audio returned for a given request must not exceed 15 seconds.

Example JSON response for a successful voice search. The comments and formatting of the JSON below is for example reasons only. Indentation spaces, smart quotes, comments, etc. are omitted from the actual response.

```

HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: audio/x-wav

Response audio payload

```

Example JSON response for a voice synthesis query failure.

```

HTTP/1.1 400 XML parser error
Content-Type: text/xml
Content-Length: 0

```

ERROR	DESCRIPTION
<b>HTTP/400 Bad Request</b>	Will be returned if a required parameter is missing, empty or null, or if the value passed to either a required or optional parameter is invalid. The "Invalid" response includes passing a string value that is longer than the allowed length. A brief description of the problematic parameter will be included.
<b>HTTP/401 Unauthorized</b>	Will be returned if the request is not authorized.
<b>HTTP/413 RequestEntityTooLarge</b>	Will be returned if the SSML input is larger than what is supported.
<b>HTTP/502 BadGateway</b>	Network related problem or a server side issue.

Example Error Response.

```

HTTP/1.0 400 Bad Request
Content-Length: XXX
Content-Type: text/plain; charset=UTF-8

Voice name not supported

```

See the [Visual C#.NET Text to Speech sample application](#) for implementation details.

## Supported Locales and Voice Fonts

Supported locales and related voice fonts include:

Locale	Gender	Service Name Mapping
ar-EG*	Female	"Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)"
de-DE	Female	"Microsoft Server Speech Text to Speech Voice (de-DE, Hedda)"
de-DE	Male	"Microsoft Server Speech Text to Speech Voice (de-DE, Stefan, Apollo)"
en-AU	Female	"Microsoft Server Speech Text to Speech Voice (en-AU, Catherine)"
en-CA	Female	"Microsoft Server Speech Text to Speech Voice (en-CA, Linda)"
en-GB	Female	"Microsoft Server Speech Text to Speech Voice (en-GB, Susan, Apollo)"
en-GB	Male	"Microsoft Server Speech Text to Speech Voice (en-GB, George, Apollo)"
en-IN	Male	"Microsoft Server Speech Text to Speech Voice (en-IN, Ravi, Apollo)"
en-US	Female	"Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)"
en-US	Male	"Microsoft Server Speech Text to Speech Voice (en-US, BenjaminRUS)"
es-ES	Female	"Microsoft Server Speech Text to Speech Voice (es-ES, Laura, Apollo)"
es-ES	Male	"Microsoft Server Speech Text to Speech Voice (es-ES, Pablo, Apollo)"
es-MX	Male	"Microsoft Server Speech Text to Speech Voice (es-MX, Raul, Apollo)"
fr-CA	Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, Caroline)"
fr-FR	Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, Julie, Apollo)"
fr-FR	Male	"Microsoft Server Speech Text to Speech Voice (fr-FR, Paul, Apollo)"

Locale	Gender	Service Name Mapping
hi-IN	Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana, Apollo)"
it-IT	Male	"Microsoft Server Speech Text to Speech Voice (it-IT, Cosimo, Apollo)"
ja-JP	Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ayumi, Apollo)"
ja-JP	Male	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ichiro, Apollo)"
ko-KR	Female	"Microsoft Server Speech Text to Speech Voice (ko-KR, HeamiRUS)"
pt-BR	Male	"Microsoft Server Speech Text to Speech Voice (pt-BR, Daniel, Apollo)"
ru-RU	Female	"Microsoft Server Speech Text to Speech Voice (ru-RU, Irina, Apollo)"
ru-RU	Male	"Microsoft Server Speech Text to Speech Voice (ru-RU, Pavel, Apollo)"
zh-CN	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, HuihuiRUS)"
zh-CN	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, Yaoyaو, Apollo)"
zh-CN	Male	"Microsoft Server Speech Text to Speech Voice (zh-CN, Kangkang, Apollo)"
zh-HK	Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, Tracy, Apollo)"
zh-HK	Male	"Microsoft Server Speech Text to Speech Voice (zh-HK, Danny, Apollo)"
zh-TW	Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, Yating, Apollo)"
zh-TW	Male	"Microsoft Server Speech Text to Speech Voice (zh-TW, Zhiwei, Apollo)"

\*ar-EG supports Modern Standard Arabic (MSA)

Post all questions and issues to the [Bing Speech Service](#) MSDN Forum, with complete detail, such as:

- An example of the full request string.
- If applicable, the full output of a failed request, which includes log IDs.
- The percentage of requests that are failing.

# Bing Spell Check API (formerly Spell Check API)

4/12/2017 • 1 min to read • [Edit Online](#)

Welcome to Bing Spell Check API which performs contextual spell checking for any text and provides inline suggestions for misspelled words.

## What is special about our 3rd generation spell-checker?

What's the difference between regular spell-checkers and Bing's 3rd generation spell-checker? Current spell-checkers rely on verifying spelling and grammar against dictionary-based rule sets which get updated and expanded periodically. In other words, the spell-checker is only as strong as the dictionary that supports it, and the editorial staff who supports the dictionary. You know this type of spell-checker from Microsoft Word and other word processors.

In contrast, the Bing team has developed a web-based spell-checker that leverages machine learning and statistical machine translation to dynamically train a constantly-evolving and highly-contextual algorithm to provide spell checking on-demand for almost any word or phrase that exists on the web.

The OneSpeller team, in partnership with Office Online, has adapted this technology to any number of word processing scenarios, texting, web form entries and search. Our spell-checker is based on a massive corpus of web searches and documents.

This spell-checker can handle any word processing scenario:

- Recognizes slang and informal language
- Recognizes common name errors in context
- Corrects word breaking issues with a single flag
- Is able to correct homophones\* in context, and other difficult to spot errors
- Supports new brands, digital entertainment and popular expressions as they emerge
- Words that sound the same but differ in meaning and spelling, for example "see" and "sea".

To help you get started with this supporting technology for all your word processing needs on PC or mobile apps, a new example application will soon be added. Check back soon. In the meantime, you can read our [Getting Started](#) guide, which describes how you can obtain your own subscription keys and start making calls to the API. If you already have a subscription, try our API Testing Console [API Testing Console](#) where you can easily craft API requests in a sandbox environment.

## Language support

Bing Spell Check API currently supports only U.S. English (en-US). Other English locales such as British English (en-GB), Canadian English (en-CA), Australian English (en-AU), New Zealand English (en-NZ), and Indian English (en-IN) plus Chinese and Spanish will be added in the near future. This page will be updated as new languages are added.

# Bing Search API Use and Display Requirements

4/12/2017 • 7 min to read • [Edit Online](#)

These use and display requirements apply to your implementation of the content and associated information (for example, relationships, metadata and other signals) available through calls to the Bing Web Search, Image Search, Video Search, and News Search APIs, Bing Spell Check and Bing Autosuggest APIs. Implementation details related to these requirements can be found in documentation for specific features and results.

## 1. BING SPELL CHECK API and BING AUTOSUGGEST API. You must not:

- copy, store, or cache any data you receive from the Bing Spell Check or Bing Autosuggest APIs; or use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.
- use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

## 2. BING WEB SEARCH, IMAGE SEARCH, NEWS SEARCH and VIDEO SEARCH APIs (the "Search APIs"):

**Definitions.** The following definitions apply in Sections 2 through 7 of these use and display requirements:

- "answer" refers to a category of results returned in a response. For example, a response from the Bing Web Search API may include answers in the categories of webpage results, image, video, and news;
- "response" means any and all answers and associated data received in response to a single call to a Search API;
- "result" refers to an item of information in an answer. For example, the set of data connected with a single news article is a result in a news answer.

**Internet search experience.** All data returned in responses may only be used in Internet search experiences. An Internet search experience means the content displayed, as applicable:

- is relevant and responsive to the end user's direct query or other indication of the user's search interest and intent (e.g., user-indicated search query);
- helps users find and navigate to the sources of data (e.g., the provided URLs are implemented as hyperlinks so the content or attribution is a clickable link conspicuously displayed with the data);
- includes multiple results for the end user to select from (e.g., several results from the news answer are displayed, or all results if fewer than several are returned);
- is limited to an amount appropriate to serve the search purpose (e.g., image thumbnails are thumbnail-sized in proportion to the user's display);
- includes visible indication to the end user that the content is Internet search results (e.g., a statement that the content is "From the web"); and
- includes any other combination of measures appropriate to ensure your use of data received from the Search APIs does not violate any applicable laws or third party rights. Please consult your legal advisors to determine what measures may be appropriate.

The only exception to the internet search experience requirement is for URL discovery as described in Section 7 (Non-display URL discovery) below.

**General.** You must not:

- copy, store, or cache any data from responses (except retention to the extent permitted by the "Continuity of Service" section below);
- modify content of results (other than to reformat them in a way that does not violate any other requirement);
- omit attribution and URLs associated with result content;
- re-order (including by omission) results displayed in an answer when an order or ranking is provided;
- display other content within any part of a response in a way that would lead an end user to believe that the other content is part of the response;
- display advertising that is not provided by Microsoft on any page that displays any part of a response;
- use data received from the Search APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

### 3. Advertising.

Advertising (whether provided by Microsoft or another provider) must not be displayed with responses (i) from the Image, News or Video Search APIs; or (ii) that are filtered or limited primarily (or solely) to image, news and/or video results from other Search APIs.

### 4. Branding.

You will attribute each response (or portion of a response) displayed to Microsoft as described in <https://go.microsoft.com/fwlink/?linkid=833278>, unless Microsoft specifies otherwise in writing for your particular use.

### 5. Transferring Responses.

If you enable a user to transfer a response to another user, such as through a messaging app or social media posting, the following apply:

- Transferred responses must:
  - Consist of content that is unmodified from the content of the responses displayed to the transferring user (formatting changes are permissible);
  - Not include any data in metadata form;
  - Display language indicating the response was obtained through an Internet search experience powered by Bing (e.g., "Powered by Bing," "Learn more about this image on Bing," or "Explore more about this image on Bing" or through the use of the Bing logo);
  - Prominently display the full query used to generate the response; and
  - Include a prominent link or similar attribution to the underlying source of the response, either directly or through bing.com or m.bing.com.
- You may not automate the transfer of responses. A transfer must be initiated by a user action clearly evidencing an intent to transfer a response.
- You may only enable a user to transfer responses obtained as a result of the transferring user's query.

## 6. Continuity of Service.

You must not copy, store or cache any data from responses. However, to enable continuity of service access and data rendering, you may retain results solely under the following conditions:

**Device.** You may enable an end user to retain results on a device for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results, provided that retained results may be used only:

- to enable the end user to access results previously returned to that end user on that device (e.g., in case of service interruption); or
- to store results returned for your proactive query personalized in anticipation of the end user's needs based on that end user's signals (e.g., in case of anticipated service interruption).

**Server.** You may retain results specific to a single end user securely on a server you control and display the retained results only:

- to enable the end user to access a historical report of results previously returned to that user in your solution, provided that the results may not be (i) retained for more than 21 days from the time of the end user's initial query and (ii) displayed in response to an end user's new or repeated query; or
- to store results returned for your proactive query personalized in anticipation of an end user's needs based on that end user's signals for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results.

Whenever retained, results for a specific user cannot be commingled with results for another user, i.e., the results of each user must be retained and delivered separately.

**General.** For all presentation of retained results, you must:

- include a clear, visible notice of the time the query was sent,
- present the user a button or similar means to re-query and obtain updated results,
- retain the Bing branding in the presentation of the results, and
- delete (and refresh with a new query if needed) the stored results within the timeframes specified.

## 7. Non-display URL discovery.

You may only use search responses in a non-internet search experience for the sole purpose of discovering URLs of sources of information responsive to a query from your user or customer. You may copy such URLs in a report or similar response you provide (i) only to that user or customer, in response to the particular query and (ii) which includes significant additional valuable content relevant to the query. The requirements in sections 2 through 6 of these use and display requirements do not apply to this non-display use, except:

- You shall not cache, copy or store any data or content from, or derived from, the search response, other than the limited URL copying described above;
- You must ensure your use of data (including the URLs) received from the Search APIs does not violate any applicable laws or third party rights; and
- You shall not use the data (including the URLs) received from the Search APIs as part of any search index or machine learning or similar algorithmic activity to create train, evaluate, or improve services which you or third parties may offer.

# Bing Video Search API

4/12/2017 • 1 min to read • [Edit Online](#)

The Bing Video Search API provides a similar (but not exact) experience to Bing.com/Videos (overview on [MSDN](#)). The Bing Video Search API lets partners send a search query to Bing and get back a list of relevant videos.

To get started, read our [Getting Started](#) guide, which describes how you can obtain your own subscription keys and start making calls to the API. If you already have a subscription, try our API Testing Console [API Testing Console](#) where you can easily craft API requests in a sandbox environment.

For information that shows you how to use the API, see [Video Guide](#).

For information about the programming elements that you'd use to request and consume the search results, see [Video Reference](#).

For additional guide and reference content that is common to all Bing APIs, such as Paging Results and Error Codes, see [Shared Guides](#) and [Shared Reference](#).

# Bing Search API Use and Display Requirements

4/12/2017 • 7 min to read • [Edit Online](#)

These use and display requirements apply to your implementation of the content and associated information (for example, relationships, metadata and other signals) available through calls to the Bing Web Search, Image Search, Video Search, and News Search APIs, Bing Spell Check and Bing Autosuggest APIs. Implementation details related to these requirements can be found in documentation for specific features and results.

## 1. BING SPELL CHECK API and BING AUTOSUGGEST API. You must not:

- copy, store, or cache any data you receive from the Bing Spell Check or Bing Autosuggest APIs; or use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.
- use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

## 2. BING WEB SEARCH, IMAGE SEARCH, NEWS SEARCH and VIDEO SEARCH APIs (the "Search APIs"):

**Definitions.** The following definitions apply in Sections 2 through 7 of these use and display requirements:

- "answer" refers to a category of results returned in a response. For example, a response from the Bing Web Search API may include answers in the categories of webpage results, image, video, and news;
- "response" means any and all answers and associated data received in response to a single call to a Search API;
- "result" refers to an item of information in an answer. For example, the set of data connected with a single news article is a result in a news answer.

**Internet search experience.** All data returned in responses may only be used in Internet search experiences. An Internet search experience means the content displayed, as applicable:

- is relevant and responsive to the end user's direct query or other indication of the user's search interest and intent (e.g., user-indicated search query);
- helps users find and navigate to the sources of data (e.g., the provided URLs are implemented as hyperlinks so the content or attribution is a clickable link conspicuously displayed with the data);
- includes multiple results for the end user to select from (e.g., several results from the news answer are displayed, or all results if fewer than several are returned);
- is limited to an amount appropriate to serve the search purpose (e.g., image thumbnails are thumbnail-sized in proportion to the user's display);
- includes visible indication to the end user that the content is Internet search results (e.g., a statement that the content is "From the web"); and
- includes any other combination of measures appropriate to ensure your use of data received from the Search APIs does not violate any applicable laws or third party rights. Please consult your legal advisors to determine what measures may be appropriate.

The only exception to the internet search experience requirement is for URL discovery as described in Section 7 (Non-display URL discovery) below.

**General.** You must not:

- copy, store, or cache any data from responses (except retention to the extent permitted by the "Continuity of Service" section below);
- modify content of results (other than to reformat them in a way that does not violate any other requirement);
- omit attribution and URLs associated with result content;
- re-order (including by omission) results displayed in an answer when an order or ranking is provided;
- display other content within any part of a response in a way that would lead an end user to believe that the other content is part of the response;
- display advertising that is not provided by Microsoft on any page that displays any part of a response;
- use data received from the Search APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

### 3. Advertising.

Advertising (whether provided by Microsoft or another provider) must not be displayed with responses (i) from the Image, News or Video Search APIs; or (ii) that are filtered or limited primarily (or solely) to image, news and/or video results from other Search APIs.

### 4. Branding.

You will attribute each response (or portion of a response) displayed to Microsoft as described in <https://go.microsoft.com/fwlink/?linkid=833278>, unless Microsoft specifies otherwise in writing for your particular use.

### 5. Transferring Responses.

If you enable a user to transfer a response to another user, such as through a messaging app or social media posting, the following apply:

- Transferred responses must:
  - Consist of content that is unmodified from the content of the responses displayed to the transferring user (formatting changes are permissible);
  - Not include any data in metadata form;
  - Display language indicating the response was obtained through an Internet search experience powered by Bing (e.g., "Powered by Bing," "Learn more about this image on Bing," or "Explore more about this image on Bing" or through the use of the Bing logo);
  - Prominently display the full query used to generate the response; and
  - Include a prominent link or similar attribution to the underlying source of the response, either directly or through bing.com or m.bing.com.
- You may not automate the transfer of responses. A transfer must be initiated by a user action clearly evidencing an intent to transfer a response.
- You may only enable a user to transfer responses obtained as a result of the transferring user's query.

## 6. Continuity of Service.

You must not copy, store or cache any data from responses. However, to enable continuity of service access and data rendering, you may retain results solely under the following conditions:

**Device.** You may enable an end user to retain results on a device for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results, provided that retained results may be used only:

- to enable the end user to access results previously returned to that end user on that device (e.g., in case of service interruption); or
- to store results returned for your proactive query personalized in anticipation of the end user's needs based on that end user's signals (e.g., in case of anticipated service interruption).

**Server.** You may retain results specific to a single end user securely on a server you control and display the retained results only:

- to enable the end user to access a historical report of results previously returned to that user in your solution, provided that the results may not be (i) retained for more than 21 days from the time of the end user's initial query and (ii) displayed in response to an end user's new or repeated query; or
- to store results returned for your proactive query personalized in anticipation of an end user's needs based on that end user's signals for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results.

Whenever retained, results for a specific user cannot be commingled with results for another user, i.e., the results of each user must be retained and delivered separately.

**General.** For all presentation of retained results, you must:

- include a clear, visible notice of the time the query was sent,
- present the user a button or similar means to re-query and obtain updated results,
- retain the Bing branding in the presentation of the results, and
- delete (and refresh with a new query if needed) the stored results within the timeframes specified.

## 7. Non-display URL discovery.

You may only use search responses in a non-internet search experience for the sole purpose of discovering URLs of sources of information responsive to a query from your user or customer. You may copy such URLs in a report or similar response you provide (i) only to that user or customer, in response to the particular query and (ii) which includes significant additional valuable content relevant to the query. The requirements in sections 2 through 6 of these use and display requirements do not apply to this non-display use, except:

- You shall not cache, copy or store any data or content from, or derived from, the search response, other than the limited URL copying described above;
- You must ensure your use of data (including the URLs) received from the Search APIs does not violate any applicable laws or third party rights; and
- You shall not use the data (including the URLs) received from the Search APIs as part of any search index or machine learning or similar algorithmic activity to create train, evaluate, or improve services which you or third parties may offer.

# Bing Web Search API

4/12/2017 • 1 min to read • [Edit Online](#)

The Bing Web Search API provides a similar (but not exact) experience to Bing.com/Search (overview on [MSDN](#)).

The Bing Web Search API lets partners send a search query to Bing and get back a list of relevant search results.

This includes webpages and may include images, videos, and more.

Typically, you'll call only the Web Search API, but if you need only images, videos or only news, you should call these APIs directly. Because calling the Image, Video, and News APIs directly can negatively impact relevance and performance, you should do so only if you need a single type of content.

If you need results from a subset of the Bing APIs, you should call the Web Search API and use the `responseFilter` query parameter to limit the results to only that content (for example, only Images and News).

Note that the Web Search API may not include all of the same functionality or data that the other APIs provide. For example, the Image API includes query parameters that let you filter the images, but you may not specify the filter parameters when you call the Search API. Also, even though the Web Search API results may not include images, the Image API could return images for the same query.

To get started, read our [Getting Started](#) guide, which describes how you can obtain your own subscription keys and start making calls to the API. If you already have a subscription, try our API Testing Console [API Testing Console](#) where you can easily craft API requests in a sandbox environment.

For information that shows you how to use the Search API, see [Search Guide](#).

For information about the programming elements that you'd use to request and consume the search results, see [Search Reference](#). Note that results will include objects defined in the Search Reference section and may include objects defined in each of the other API reference sections (for example, [Image Reference](#)), if relevant data exists for each.

For additional guide and reference content that is common to all Bing APIs, such as Paging Results and Error Codes, see [Shared Guides](#) and [Shared Reference](#).

# Bing Search API Use and Display Requirements

4/12/2017 • 7 min to read • [Edit Online](#)

These use and display requirements apply to your implementation of the content and associated information (for example, relationships, metadata and other signals) available through calls to the Bing Web Search, Image Search, Video Search, and News Search APIs, Bing Spell Check and Bing Autosuggest APIs. Implementation details related to these requirements can be found in documentation for specific features and results.

## 1. BING SPELL CHECK API and BING AUTOSUGGEST API. You must not:

- copy, store, or cache any data you receive from the Bing Spell Check or Bing Autosuggest APIs; or use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.
- use data you receive from the Bing Spell Check or Bing Autosuggest APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

## 2. BING WEB SEARCH, IMAGE SEARCH, NEWS SEARCH and VIDEO SEARCH APIs (the "Search APIs"):

**Definitions.** The following definitions apply in Sections 2 through 7 of these use and display requirements:

- "answer" refers to a category of results returned in a response. For example, a response from the Bing Web Search API may include answers in the categories of webpage results, image, video, and news;
- "response" means any and all answers and associated data received in response to a single call to a Search API;
- "result" refers to an item of information in an answer. For example, the set of data connected with a single news article is a result in a news answer.

**Internet search experience.** All data returned in responses may only be used in Internet search experiences. An Internet search experience means the content displayed, as applicable:

- is relevant and responsive to the end user's direct query or other indication of the user's search interest and intent (e.g., user-indicated search query);
- helps users find and navigate to the sources of data (e.g., the provided URLs are implemented as hyperlinks so the content or attribution is a clickable link conspicuously displayed with the data);
- includes multiple results for the end user to select from (e.g., several results from the news answer are displayed, or all results if fewer than several are returned);
- is limited to an amount appropriate to serve the search purpose (e.g., image thumbnails are thumbnail-sized in proportion to the user's display);
- includes visible indication to the end user that the content is Internet search results (e.g., a statement that the content is "From the web"); and
- includes any other combination of measures appropriate to ensure your use of data received from the Search APIs does not violate any applicable laws or third party rights. Please consult your legal advisors to determine what measures may be appropriate.

The only exception to the internet search experience requirement is for URL discovery as described in Section 7 (Non-display URL discovery) below.

**General.** You must not:

- copy, store, or cache any data from responses (except retention to the extent permitted by the "Continuity of Service" section below);
- modify content of results (other than to reformat them in a way that does not violate any other requirement);
- omit attribution and URLs associated with result content;
- re-order (including by omission) results displayed in an answer when an order or ranking is provided;
- display other content within any part of a response in a way that would lead an end user to believe that the other content is part of the response;
- display advertising that is not provided by Microsoft on any page that displays any part of a response;
- use data received from the Search APIs as part of any machine learning or similar algorithmic activity to train, evaluate, or improve new or existing services which you or third parties may offer.

### 3. Advertising.

Advertising (whether provided by Microsoft or another provider) must not be displayed with responses (i) from the Image, News or Video Search APIs; or (ii) that are filtered or limited primarily (or solely) to image, news and/or video results from other Search APIs.

### 4. Branding.

You will attribute each response (or portion of a response) displayed to Microsoft as described in <https://go.microsoft.com/fwlink/?linkid=833278>, unless Microsoft specifies otherwise in writing for your particular use.

### 5. Transferring Responses.

If you enable a user to transfer a response to another user, such as through a messaging app or social media posting, the following apply:

- Transferred responses must:
  - Consist of content that is unmodified from the content of the responses displayed to the transferring user (formatting changes are permissible);
  - Not include any data in metadata form;
  - Display language indicating the response was obtained through an Internet search experience powered by Bing (e.g., "Powered by Bing," "Learn more about this image on Bing," or "Explore more about this image on Bing" or through the use of the Bing logo);
  - Prominently display the full query used to generate the response; and
  - Include a prominent link or similar attribution to the underlying source of the response, either directly or through bing.com or m.bing.com.
- You may not automate the transfer of responses. A transfer must be initiated by a user action clearly evidencing an intent to transfer a response.
- You may only enable a user to transfer responses obtained as a result of the transferring user's query.

## 6. Continuity of Service.

You must not copy, store or cache any data from responses. However, to enable continuity of service access and data rendering, you may retain results solely under the following conditions:

**Device.** You may enable an end user to retain results on a device for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results, provided that retained results may be used only:

- to enable the end user to access results previously returned to that end user on that device (e.g., in case of service interruption); or
- to store results returned for your proactive query personalized in anticipation of the end user's needs based on that end user's signals (e.g., in case of anticipated service interruption).

**Server.** You may retain results specific to a single end user securely on a server you control and display the retained results only:

- to enable the end user to access a historical report of results previously returned to that user in your solution, provided that the results may not be (i) retained for more than 21 days from the time of the end user's initial query and (ii) displayed in response to an end user's new or repeated query; or
- to store results returned for your proactive query personalized in anticipation of an end user's needs based on that end user's signals for the lesser of (i) 24 hours from the time of the query or (ii) until an end user submits another query for updated results.

Whenever retained, results for a specific user cannot be commingled with results for another user, i.e., the results of each user must be retained and delivered separately.

**General.** For all presentation of retained results, you must:

- include a clear, visible notice of the time the query was sent,
- present the user a button or similar means to re-query and obtain updated results,
- retain the Bing branding in the presentation of the results, and
- delete (and refresh with a new query if needed) the stored results within the timeframes specified.

## 7. Non-display URL discovery.

You may only use search responses in a non-internet search experience for the sole purpose of discovering URLs of sources of information responsive to a query from your user or customer. You may copy such URLs in a report or similar response you provide (i) only to that user or customer, in response to the particular query and (ii) which includes significant additional valuable content relevant to the query. The requirements in sections 2 through 6 of these use and display requirements do not apply to this non-display use, except:

- You shall not cache, copy or store any data or content from, or derived from, the search response, other than the limited URL copying described above;
- You must ensure your use of data (including the URLs) received from the Search APIs does not violate any applicable laws or third party rights; and
- You shall not use the data (including the URLs) received from the Search APIs as part of any search index or machine learning or similar algorithmic activity to create train, evaluate, or improve services which you or third parties may offer.

# Computer Vision API Version 1.0

4/18/2017 • 9 min to read • [Edit Online](#)

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 users 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 handwritten 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



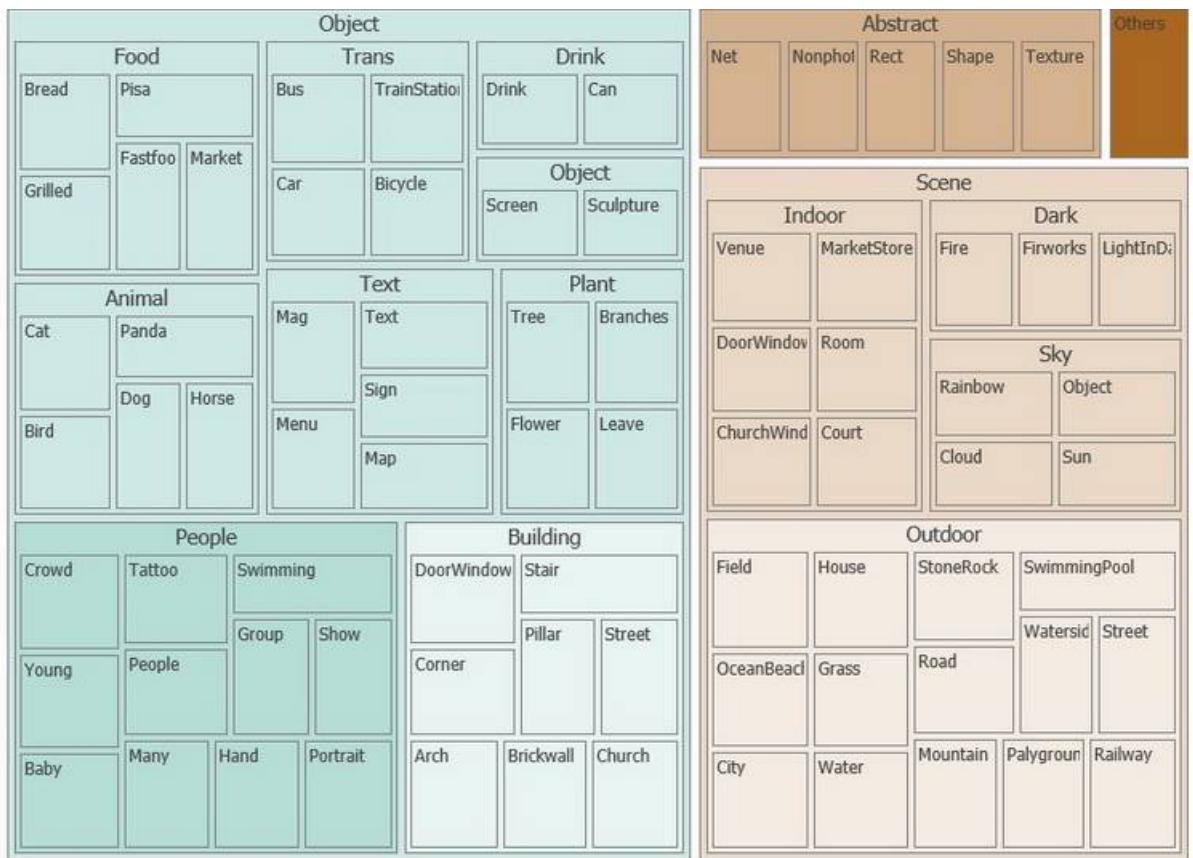
```
Returned Json
{
  "tags": [
    {
      "name": "grass",
      "confidence": 0.999999761581421
    },
    {
      "name": "outdoor",
      "confidence": 0.999970674514771
    },
    {
      "name": "sky",
      "confidence": 999289751052856
    },
    {
      "name": "building",
      "confidence": 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. It 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 as such 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

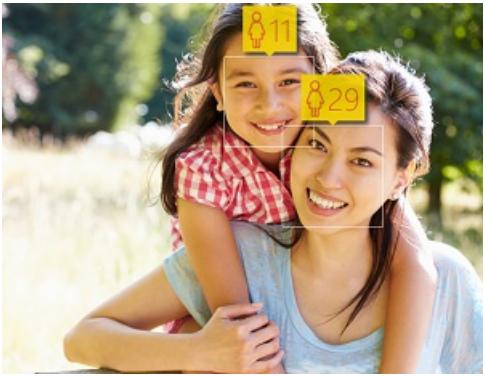
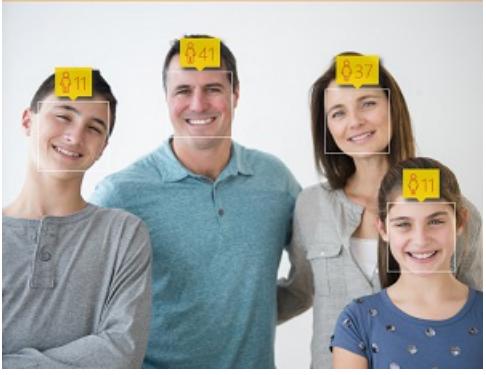
Detects whether an image is a line drawing or not.

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": 28, "gender": "Female", "faceRectangle": { "left": 447, "top": 195, "width": 162, "height": 162 } }, { "age": 10, "gender": "Male", "faceRectangle": { "left": 355, "top": 87, "width": 143, "height": 143 } } ]
	[ { "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. For this option, if you know which model you want to use, you specify the model's name, and 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 twelve 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

IMAGE	FOREGROUND	BACKGROUND	COLORS
	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
	#484B83

### Black & White

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

IMAGE	RESPONSE
	True
	False

## Flagging Adult Content

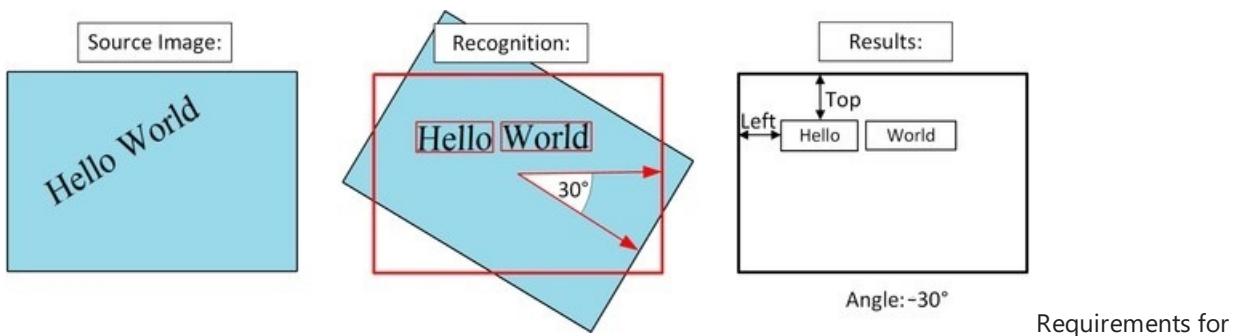
Among the various visual categories is the adult and racy group, which enables detection of adult materials and restricts the display of images containing sexual content. The filter for adult and racy content detection can be set on a sliding scale to accommodate the user's preference.

## 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 21 languages. These languages are: Chinese Simplified, Chinese Traditional, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Russian, 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 32000 x 32000 pixels.
- The image cannot be bigger than 100 megapixels.

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.

- 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

## Recognize Handwritten Text

This technology allows you to detect and extract handwritten text from notes, letters, essays, whiteboards, forms, etc. It works with different surfaces and backgrounds, such as white paper, yellow sticky notes, and whiteboards.

Handwritten text recognition saves time and effort and can make you more productive by allowing you to take images of text, rather than having to transcribe it. It makes it possible to digitize notes. This digitization allows you to implement quick and easy search. It also reduces paper clutter.

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.



<https://oxfordportal.blob.core>



# Obtaining Subscription Keys

4/18/2017 • 1 min to read • [Edit Online](#)

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

Free subscriptions that are not used for 90 days consecutively may expire. However, if your subscription expires you can request new keys.

## Related Links:

- [Pricing Options for Microsoft Cognitive APIs](#)

# How to Call Computer Vision API

4/12/2017 • 5 min 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 retune.

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/v1.0/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/v1.0/analyze?  
visualFeatures=Description&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/v1.0/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/v1.0/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/v1.0/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/v1.0/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/v1.0/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 is 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

4/12/2017 • 7 min 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 uses 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();

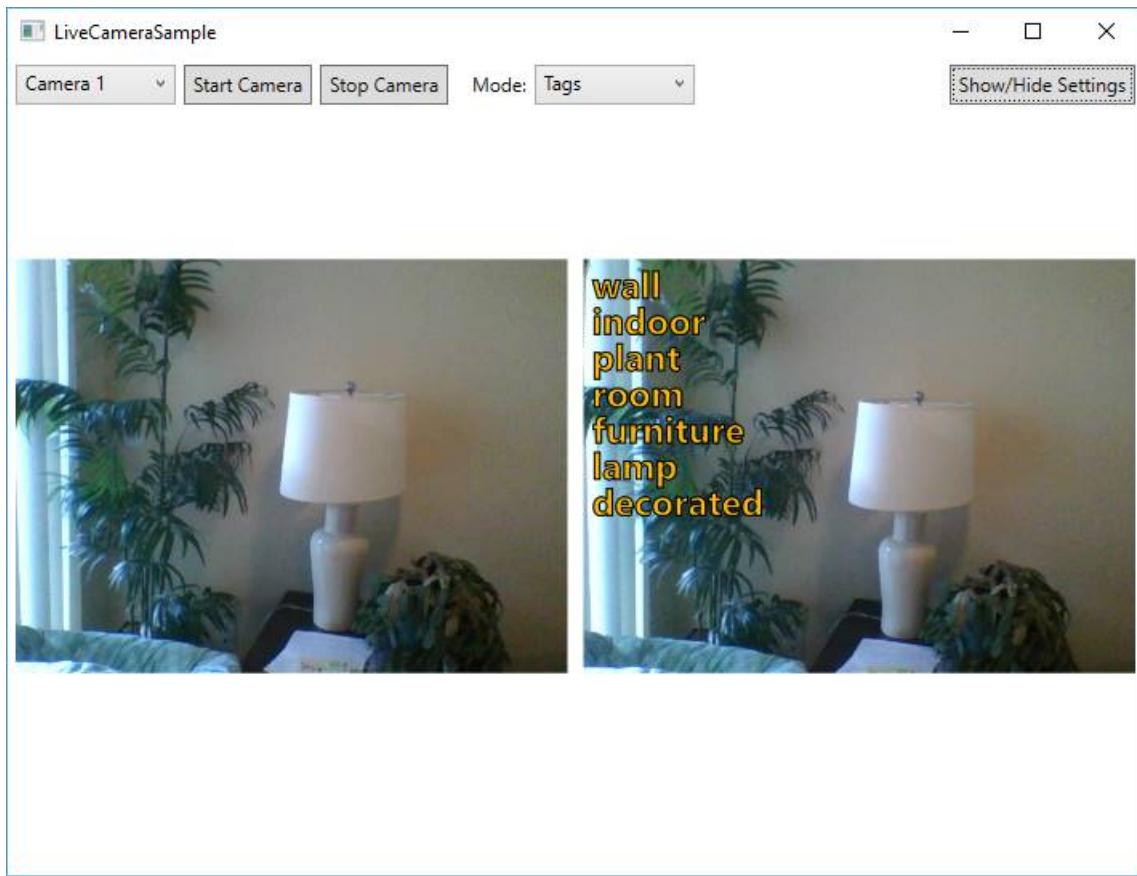
            // Wait for keypress to stop
            Console.WriteLine("Press any key to stop...");
            Console.ReadKey();

            // 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 [microsoft.com/cognitive](https://microsoft.com/cognitive). 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.

To report abuse of the Microsoft Cognitive Services to Microsoft, please visit the [Microsoft Cognitive Services website](#), and use the "Report Abuse" link at the bottom of the page to contact Microsoft. For more information about Microsoft privacy policies please see their privacy statement here: <https://go.microsoft.com/fwlink/?LinkId=521839>.

## 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](#).

# Computer Vision cURL Quick Starts

4/18/2017 • 2 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with cURL to accomplish the following tasks:

- [Analyze an image](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using cURL

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:

- The category defined in this [taxonomy](#).
- 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 (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an Image curl Example Request

```
@ECHO OFF

curl -v -X POST "https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?
visualFeatures=Categories&details={string}&language=en"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"
```

### Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "people_",
      "score": 0.83984375,
      "detail": {
        "celebrities": [
          {
            "name": "Satya Nadella",
            "faceRectangle": {
              "top": 120,
              "left": 150,
              "width": 100,
              "height": 100
            }
          }
        ]
      }
    }
  ]
}
```

```
        "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
}
}
```

## Get a Thumbnail with Computer Vision API Using curl

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail curl Example Request

```
@ECHO OFF

curl -v -X POST "https://westus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail?width={number}&height={number}&smartCropping=true"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"
```

### Get a Thumbnail Response

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

## Optical Character Recognition (OCR) with Computer Vision API Using curl

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-readable character stream.

### OCR curl Example Request

```
@ECHO OFF

curl -v -X POST "https://westus.api.cognitive.microsoft.com/vision/v1.0/ocr?language=unk&detectOrientation=true"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"
```

### OCR Example Response

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

```
{  
    "language": "en",  
    "textAngle": -2.000000000000338,  
    "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"  
                        }  
                    ]  
                },  
                {  
                    "boundingBox": "801,564,153,73",  
                    "text": "IS"  
                }  
            ]  
        }  
    ]  
}
```

# Computer Vision C# Quick Starts

4/18/2017 • 8 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with C# to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract printed text from an image](#)
- [Detect and extract handwritten text from an image](#)

## Prerequisites

- Get the Microsoft Computer Vision API Windows SDK [here](#).
- To use the Computer Vision API, you need a subscription key. You can get free subscription keys [here](#).

## Analyze an Image With Computer Vision API Using C#

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:

- The category defined in this [taxonomy](#).
- 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.
- Does the image contain adult or sexually suggestive content?

### Analyze an Image C# Example Request

```

using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter image file path: ");
            string imagePath = Console.ReadLine();

            MakeAnalysisRequest(imagePath);

            Console.WriteLine("\n\n\nHit ENTER to exit...");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void MakeAnalysisRequest(string imagePath)
        {
            var client = new HttpClient();

            // Request headers - replace this example key with your valid subscription key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request parameters. A third optional parameter is "details".
            string requestParameters = "visualFeatures=Categories&language=en";
            string uri = "https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?" + requestParameters;
            Console.WriteLine(uri);

            HttpResponseMessage response;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);

            using (var 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");
                response = await client.PostAsync(uri, content);
            }
        }
    }
}

```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "background_",
      "score": 0.00390625
    },
    {
      "name": "text_",
      "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
}
}
```

## Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### Landmark C# Example Request

```

using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter image file path: ");
            string imagePath = Console.ReadLine();

            MakeAnalysisRequest(imagePath);

            Console.WriteLine("\n\nHit ENTER to exit...\n");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void MakeAnalysisRequest(string imagePath)
        {
            var client = new HttpClient();

            // Request headers. Replace the second parameter with a valid subscription key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request parameters. Change "landmarks" to "celebrities" on requestParameters and uri to use the
            // Celebrities model.
            string requestParameters = "model=landmarks";
            string uri = "https://westus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze?" +
            requestParameters;
            Console.WriteLine(uri);

            HttpResponseMessage response;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);

            using (var 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");
                response = await client.PostAsync(uri, content);
                string contentString = await response.Content.ReadAsStringAsync();
                Console.WriteLine("Response:\n");
                Console.WriteLine(contentString);
            }
        }
    }
}

```

## Landmark Example Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{  
  "requestId": "b15f13a4-77d9-4fab-a701-7ad65bcdcaed",  
  "metadata": {  
    "width": 1024,  
    "height": 680,  
    "format": "Jpeg"  
  },  
  "result": {  
    "landmarks": [  
      {  
        "name": "Space Needle",  
        "confidence": 0.9448209  
      }  
    ]  
  }  
}
```

## Get a Thumbnail with Computer Vision API Using C#

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire. You can even pick an aspect ratio that differs from the aspect ratio of the input image.

### Get a Thumbnail C# Example Request

```

using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter image file path: ");
            string imagePath = Console.ReadLine();

            MakeThumbnailRequest(imagePath);

            Console.WriteLine("\n\n\nHit ENTER to exit...");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void MakeThumbnailRequest(string imagePath)
        {
            var client = new HttpClient();

            // Request headers - replace this example key with your valid subscription key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request parameters and URI.
            string requestParameters = "width=200&height=150&smartCropping=true";
            string uri = "https://westus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail?" +
            requestParameters;

            HttpResponseMessage response;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);

            using (var 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");
                response = await client.PostAsync(uri, content);
            }
        }
    }
}

```

## Get a 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.

## Optical Character Recognition (OCR) with Computer Vision API Using C#

Use the [Optical Character Recognition \(OCR\) method](#) to detect printed text in an image and extract recognized

characters into a machine-readable character stream.

## OCR C# Example Request

```
using System;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter image file path: ");
            string imagePath = Console.ReadLine();

            MakeOCRRequest(imagePath);

            Console.WriteLine("\n\n\nHit ENTER to exit...");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void MakeOCRRequest(string imagePath)
        {
            var client = new HttpClient();

            // Request headers - replace this example key with your valid subscription key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request parameters and URI
            string requestParameters = "language=unk&detectOrientation =true";
            string uri = "https://westus.api.cognitive.microsoft.com/vision/v1.0/ocr?" + requestParameters;

            HttpResponseMessage response;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);

            using (var 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");
                response = await client.PostAsync(uri, content);
            }
        }
    }
}
```

## OCR Example Response

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

```
{
    "language": "en",
    "textAngle": -2.000000000000338,
    "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"
                        }
                    ]
                }
            ]
        }
    ]
}
```

## Text Recognition with Computer Vision API Using C#

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-readable character stream.

## Handwriting Recognition C# Example

```
using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.Write("Enter image file path: ");
            string imagePath = Console.ReadLine();

            ReadHandwrittenText(imagePath);

            Console.WriteLine("\n\n\nHit ENTER to exit...");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void ReadHandwrittenText(string imagePath)
        {
            var client = new HttpClient();

            // Request headers - replace this example key with your valid subscription key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request parameters and URI. Set "handwriting" to false for printed text.
            string requestParameter = "handwriting=true";
            string uri = "https://westus.api.cognitive.microsoft.com/vision/v1.0/recognizeText?" +
requestParameter;

            HttpResponseMessage response = null;
            IEnumerable<string> responseValues = null;
            string operationLocation = null;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);
            var content = new ByteArrayContent(byteData);

            // This example uses content type "application/octet-stream".
            // You can also use "application/json" and specify an image URL.
            content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

            try {
                response = await client.PostAsync(uri, content);
                responseValues = response.Headers.GetValues("Operation-Location");
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            foreach (var value in responseValues)
```

```
    // This value is the URI where you can get the text recognition operation result.
    operationLocation = value;
    Console.WriteLine(operationLocation);
    break;
}

try
{
    // 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.
    response = await client.GetAsync(operationLocation);

    // And now you can see the response in in JSON:
    Console.WriteLine(await response.Content.ReadAsStringAsync());
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}

}

}
```

# Computer Vision Java Quick Starts

4/18/2017 • 7 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using Java and the Computer Vision API to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract printed text from an image](#)
- [Detect and extract handwritten text from an image](#)

## Prerequisites

- Get the Microsoft Computer Vision Android SDK [here](#).
- To use the Computer Vision API, you need a subscription key. You can get free subscription keys [here](#).

## Analyze an Image With Computer Vision API Using Java

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:

- The category defined in this [taxonomy](#).
- 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.
- Does the image contain adult or sexually suggestive content?

### Analyze an Image Java Example Request

```

// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
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.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpclient = new DefaultHttpClient();

        try
        {
            URIBuilder builder = new
URIBuilder("https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze");

            builder.setParameter("visualFeatures", "Categories");
            builder.setParameter("details", "Celebrities");
            builder.setParameter("language", "en");

            URI uri = builder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers - replace this example key with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. Replace the example URL with the URL for the JPEG image of a celebrity.
            StringEntity reqEntity = new StringEntity("{\"url\":\"http://example.com/images/test.jpg\"}");
            request.setEntity(reqEntity);

            HttpResponse response = httpclient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                System.out.println(EntityUtils.toString(entity));
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

## Analyze an Image Response

A successful response is returned in JSON. The following example shows a successful response:

```
{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "people_",
      "score": 0.83984375,
      "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
    }
}
```

## Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### **Landmark Java Example Request**

```

// This sample uses the Apache HTTP client (org.apache.httpcomponents:httpclient:4.2.4)
// and org.json (org.json:20160810).

import java.net.URI;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
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.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            // Change "landmarks" to "celebrities" in the url to use the Celebrities model.
            URIBuilder uriBuilder = new
URIBuilder("https://westus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze");

            // Change "landmarks" to "celebrities" to use the Celebrities model.
            uriBuilder.setParameter("model", "landmarks");

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

            // Request headers. Replace the example key below with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. Replace the example URL with the URL of a JPEG image containing text.
            StringEntity requestEntity = new StringEntity(
                "{\"url\":\"https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg\"}");
            request.setEntity(requestEntity);

            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                // Output the JSON response
                String jsonString = EntityUtils.toString(entity);
                JSONObject json = new JSONObject(jsonString);
                System.out.println("REST Response:");
                System.out.println(json.toString(2));
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

## Landmark Example Response

A successful response is returned in JSON. Following is an example of a successful response:

```

REST Response:
{
  "result": {"landmarks": [{"confidence": 0.9998178, "name": "Space Needle"}]}, 
  "metadata": {"width": 2096, "format": "Jpeg", "height": 4132}, 
  "requestId": "7d0d00da-ac37-44ba-ad77-050e11a5ee05"
}

```

## Get a Thumbnail with Computer Vision API Using Java

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire. The aspect ratio you set for the thumbnail can be different from the aspect ratio of the input image.

### Get a Thumbnail Java Example Request

```

// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-ga/)
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.HttpClient;
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.DefaultHttpClient;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            URIBuilder uriBuilder = new URIBuilder("https://westus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail");

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

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

            // Request headers - replace this example key with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. Replace the example URL with the URL for the JPEG image of a person.
            StringEntity requestEntity = new StringEntity("{\"url\":\"http://example.com/images/test.jpg\"}");
            request.setEntity(requestEntity);

            HttpResponse response = httpClient.execute(request);
            System.out.println(response);
        }
    }
}

```

```

        // Display the thumbnail.
        HttpEntity httpEntity = response.getEntity();
        displayImage(httpEntity.getContent());
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
}

private static void displayImage(InputStream inputStream)
{
    try {
        BufferedImage bufferedImage = ImageIO.read(inputStream);

        ImageIcon imageIcon = new ImageIcon(bufferedImage);

        JLabel jLabel = new JLabel();
        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 a 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.

## Optical Character Recognition (OCR) with Computer Vision API Using Java

Use the [Optical Character Recognition \(OCR\) method](#) to detect printed text in an image and extract recognized characters into a machine-readable character stream.

### OCR Java Example Request

```

// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
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.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            URIBuilder uriBuilder = new
URIBuilder("https://westus.api.cognitive.microsoft.com/vision/v1.0/ocr");

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

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

            // Request headers - replace this example key with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. Replace the example URL with the URL of a JPEG image containing text.
            StringEntity requestEntity = new StringEntity("{\"url\":\"http://example.com/images/test.jpg\"}");
            request.setEntity(requestEntity);

            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                System.out.println(EntityUtils.toString(entity));
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

## OCR Example Response

Upon success, the OCR results returned include the detected text and bounding boxes for regions, lines, and words.

```
{  
    "language": "en",  
    "textAngle": -2.000000000000338,  
    "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"  
                        }  
                    ]  
                },  
                {  
                    "boundingBox": "801,564,153,73",  
                    "text": "IS"  
                }  
            ]  
        }  
    ]  
}
```

## Text Recognition with Computer Vision API Using Java

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-readable character stream.

## Handwriting Recognition Java Example

```
// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-4.5.x/)
import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.apache.http.Header;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient textClient = new DefaultHttpClient();
        HttpClient resultClient = new DefaultHttpClient();

        // Replace this example key with your valid subscription key.
        String subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

        try
        {
            // For printed text, set "handwriting" to false.
            URI uri = new URI("https://westus.api.cognitive.microsoft.com/vision/v1.0/recognizeText?handwriting=true");
            HttpPost textRequest = new HttpPost(uri);

            // Request headers. Another valid content type is "application/octet-stream".
            textRequest.setHeader("Content-Type", "application/json");
            textRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body. Replace the example URL with the URL of a JPEG image containing handwriting.
            StringEntity requestEntity = new StringEntity("{\"url\":\"http://example.com/images/test.jpg\"}");
            textRequest.setEntity(requestEntity);

            HttpResponse textResponse = textClient.execute(textRequest);
            String operationLocation = null;

            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;
                }
            }

            System.out.println(operationLocation);

            HttpGet resultRequest = new HttpGet(operationLocation);
            resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // 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.
            HttpResponse resultResponse = resultClient.execute(resultRequest);
            System.out.print("Text recognition result response: ");
            System.out.println(resultResponse);
        }
    }
}
```

```
        }
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
}
```

# Computer Vision JavaScript Quick Starts

4/18/2017 • 4 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using JavaScript and the Computer Vision API to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using JavaScript

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:

- The category defined in this [taxonomy](#).
- 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 (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an Image JavaScript Example Request

```

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

<script type="text/javascript">
$(function() {
    var params = {
        // Request parameters
        "visualFeatures": "Categories",
        "details": "{string}",
        "language": "en",
    };

    $.ajax({
        url: "https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?" + $.param(params),
        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","{subscription key}");
        },
        type: "POST",
        // Request body
        data: "{body}",
    })
    .done(function(data) {
        alert("success");
    })
    .fail(function() {
        alert("error");
    });
});
</script>
</body>
</html>

```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "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
}
}
```

## Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### Landmark JavaScript Example Request

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript 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() {
    var params = {
        // Request parameters
        "model": "landmarks", // Use "model": "celebrities" to use the Celebrities model.
    };

    $.ajax({
        // Change "landmarks" to "celebrities" in the url to use the Celebrities model.
        url: "https://westus.api.cognitive.microsoft.com/v1.0/models/landmarks/analyze?" +
        $.param(params),

        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type", "application/json");

            // Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");
        },

        type: "POST",

        // Request body
        data: '{"url": "https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg"}',
    })

    .done(function(data) {
        $("#responseTextArea").val(JSON.stringify(data, null, 2));
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : jQuery.parseJSON(jqXHR.responseText).message;
        alert(errorString);
    });
});
</script>
REST response:
<br><br>
<textarea id="responseTextArea" class="UIInput" cols="120" rows="32"></textarea>
</body>
</html>
```

## Landmark Example Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{  
  "requestId": "e0970003-1cb7-4ac6-b0d4-f36a1914bf4e",  
  "metadata": {  
    "width": 2096,  
    "height": 4132,  
    "format": "Jpeg"  
  },  
  "result": {  
    "landmarks": [  
      {  
        "name": "Space Needle",  
        "confidence": 0.9998178  
      }  
    ]  
  }  
}
```

## Get a Thumbnail with Computer Vision API Using JavaScript

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail JavaScript Example Request

```

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

<script type="text/javascript">
$(function() {
    var params = {
        // Request parameters
        "width": "{number}",
        "height": "{number}",
        "smartCropping": "true",
    };

    $.ajax({
        url: "https://westus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail?" + $.param(params),
        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","{subscription key}");
        },
        type: "POST",
        // Request body
        data: "{body}",
    })
    .done(function(data) {
        alert("success");
    })
    .fail(function() {
        alert("error");
    });
});
</script>
</body>
</html>

```

## Get a Thumbnail Response

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

## Optical Character Recognition (OCR) with Computer Vision API Using JavaScript

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-readable character stream.

### OCR JavaScript Example Request

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

<script type="text/javascript">
$(function() {
    var params = {
        // Request parameters
        "language": "unk",
        "detectOrientation ": "true",
    };

    $.ajax({
        url: "https://westus.api.cognitive.microsoft.com/vision/v1.0/ocr?" + $.param(params),
        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","{subscription key}");
        },
        type: "POST",
        // Request body
        data: "{body}",
    })
    .done(function(data) {
        alert("success");
    })
    .fail(function() {
        alert("error");
    });
});
});
</script>
</body>
</html>
```

## OCR Example Response

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

```
{  
    "language": "en",  
    "textAngle": -2.000000000000338,  
    "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"  
                        }  
                    ]  
                },  
                {  
                    "boundingBox": "801,564,153,73",  
                    "text": "IS"  
                }  
            ]  
        }  
    ]  
}
```

# Computer Vision PHP Quick Starts

4/18/2017 • 4 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using PHP and the Computer Vision API to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using PHP

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:

- The category defined in this [taxonomy](#).
- 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 (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an Image PHP Example Request

```

<?php
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)
require_once 'HTTP/Request2.php';

$request = new Http_Request2('https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze');
$url = $request->getUrl();

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

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

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

// Request body
$request->setBody("{body}");

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "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
    }
}
```

## Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### **Landmark PHP Example Request**

```

<html>
<head>
    <title>PHP Sample</title>
</head>
<body>
<?php
// This sample uses PEAR (https://pear.php.net/package/HTTP_Request2/download)
require_once 'HTTP/Request2.php';

// Change "landmarks" to "celebrities" in the url to use the Celebrities model.
$request = new
Http_Request2('https://westus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => '13hc77781f7e4b19b5fcdd72a8df7156',
);
$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'model' => 'landmarks',    // Use 'model' => 'celebrities' to use the Celebrities model.
);
$url->setQueryVariables($parameters);

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

// Request body
$body = json_encode(array(
    // Request body parameters
    'url' => 'https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg',
));
$request->setBody($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>

```

## Landmark Example Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{  
    "requestId": "0663b074-8eb3-4fab-a72e-4c31a49bd22e",  
    "metadata": {  
        "width": 2096,  
        "height": 4132,  
        "format": "Jpeg"  
    },  
    "result": {  
        "landmarks": [  
            {  
                "name": "Space Needle",  
                "confidence": 0.9998178  
            }  
        ]  
    }  
}
```

## Get a Thumbnail with Computer Vision API Using PHP

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail PHP Example Request

```

<?php

// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

require_once 'HTTP/Request2.php';

$request = new Http_Request2('https://westus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => '{subscription key}',
);

$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'width' => '{number}',
    'height' => '{number}',
    'smartCropping' => 'true',
);

$url->setQueryVariables($parameters);

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

// Request body
$request->setBody("{body}");

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

## Get a Thumbnail Response

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

## Optical Character Recognition (OCR) with Computer Vision API Using PHP

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-readable character stream.

### OCR PHP Example Request

```
<?php

// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

require_once 'HTTP/Request2.php';

$request = new Http_Request2('https://westus.api.cognitive.microsoft.com/vision/v1.0/ocr');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => '{subscription key}',
);

$request->setHeader($headers);

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

$url->setQueryVariables($parameters);

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

// Request body
$request->setBody("{body}");

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>
```

## OCR Example Response

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

```
{  
    "language": "en",  
    "textAngle": -2.000000000000338,  
    "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"  
                        }  
                    ]  
                },  
                {  
                    "boundingBox": "801,564,153,73",  
                    "text": "IS"  
                }  
            ]  
        }  
    ]  
}
```

# Computer Vision Python Quick Starts

4/18/2017 • 9 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with Python to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract printed text from an image](#)
- [Detect and extract handwritten text from an image](#)

To use the Computer Vision API, you need a subscription key. You can get free subscription keys [here](#).

## Analyze an Image With Computer Vision API Using Python

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:

- The category defined in this [taxonomy](#).
- 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.
- Does the image contain adult or sexually suggestive content?

### Analyze an Image Python Example Request

```

##### Python 2.7 #####
import httplib, urllib, base64

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.urlencode({
    # Request parameters. All of them are optional.
    'visualFeatures': 'Categories',
    'details': 'Celebrities',
    'language': 'en',
})

# Replace the three dots below with the URL of a JPEG image of a celebrity.
body = {"url": "..."}"

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/analyze?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.parse.urlencode({
    # Request parameters. All of them are optional.
    'visualFeatures': 'Categories',
    'details': 'Celebrities',
    'language': 'en',
})

# Replace the three dots below with the URL of a JPEG image of a celebrity.
body = {"url": "..."}"

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/analyze?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))
#####

```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
    "categories": [
        {
            "name": "celebrity"
        }
    ],
    "details": [
        {
            "name": "celebrity"
        }
    ],
    "language": "en"
}
```

```
"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
}
}
```

## Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### Landmark Python Example Request

```

##### Python 2.7 #####
import httplib, urllib, base64, json

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.urlencode({
    # Request parameters. Use "model": "celebrities" to use the Celebrity model.
    'model': 'landmarks',
})

# The URL of a JPEG image containing text.
body = "{url:'https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg'}"

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    # Change "landmarks" to "celebrities" in the url to use the Celebrity model.
    conn.request("POST", "/vision/v1.0/models/landmarks/analyze?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    # 'data' contains the JSON data. The following formats the JSON data for display.
    parsed = json.loads(data)
    print ("REST Response:")
    print (json.dumps(parsed, sort_keys=True, indent=2))
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64, json

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.parse.urlencode({
    # Request parameters. Use "model": "celebrities" to use the Celebrity model.
    'model': 'landmarks',
})

# The URL of a JPEG image containing text.
body = "{url:'https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg'}"

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/models/landmarks/analyze?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    # 'data' contains the JSON data. The following formats the JSON data for display.
    encoding = response.headers.get_content_charset()
    parsed = json.loads(data.decode(encoding))
    print ("REST Response:")
    print (json.dumps(parsed, sort_keys=True, indent=2))
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####

```

## Landmark Example Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{  
    "metadata": {  
        "format": "Jpeg",  
        "height": 4132,  
        "width": 2096  
    },  
    "requestId": "d08a914a-0fbb-4695-9a2e-c93791865436",  
    "result": {  
        "landmarks": [  
            {  
                "confidence": 0.9998178,  
                "name": "Space Needle"  
            }  
        ]  
    }  
}
```

## Get a Thumbnail with Computer Vision API Using Python

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire. The aspect ratio you set for the thumbnail can be different from the aspect ratio of the input image.

### Get a Thumbnail Python Example Request

```

##### Python 2.7 #####
import httplib, urllib, base64

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.urlencode({
    # Request parameters. The smartCropping flag is optional.
    'width': '150',
    'height': '100',
    'smartCropping': 'true',
})

# Replace the three dots below with the URL of the JPEG image for which you want a thumbnail.
body = {"url": "..."}"

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/generateThumbnail?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.parse.urlencode({
    # Request parameters. The smartCropping flag is optional.
    'width': '150',
    'height': '100',
    'smartCropping': 'true',
})

# Replace the three dots below with the URL of the JPEG image for which you want a thumbnail.
body = {"url": "..."}"

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/generateThumbnail?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))
#####

```

## Get a 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.

# Optical Character Recognition (OCR) with Computer Vision API Using Python

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-readable character stream.

## OCR Python Example Request

```

##### Python 2.7 #####
import httplib, urllib, base64

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.urlencode({
    # Request parameters. The language setting "unk" means automatically detect the language.
    'language': 'unk',
    'detectOrientation ': 'true',
})

# Replace the three dots below with the URL of a JPEG image containing text.
body = {"url": "..."}"

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/ocr?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64

headers = {
    # Request headers. Replace the key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.parse.urlencode({
    # Request parameters. The language setting "unk" means automatically detect the language.
    'language': 'unk',
    'detectOrientation ': 'true',
})

# Replace the three dots below with the URL of a JPEG image containing text.
body = {"url": "..."}"

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/vision/v1.0/ocr?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))
#####

```

## OCR Example Response

Upon success, the OCR results include the text from the image. They also include bounding boxes for regions, lines, and words.

```
{  
    "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"  
                        }  
                    ]  
                },  
                {  
                    "boundingBox": "801,564,153,73",  
                    "text": "IS"  
                }  
            ]  
        }  
    ]  
}
```

## Text recognition with Computer Vision API Using Python

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-readable character stream.

## Handwriting Recognition Python Example

```
##### Python 2.7 #####
import httplib, urllib, base64, time

headers = {
    # Request headers - replace this example key with your valid subscription key.
    # Another valid content type is "application/octet-stream".
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

# Replace the three dots below with the URL of a JPEG image containing text.
body = {"url": "..."} 

serviceUrl = 'westus.api.cognitive.microsoft.com'

# For printed text, set "handwriting" to false.
params = urllib.urlencode({'handwriting' : 'true'})

try:
    conn = httplib.HTTPSConnection(serviceUrl)
    conn.request("POST", "/vision/v1.0/RecognizeText?%s" % params, body, headers)
    response = conn.getresponse()

    # This is the URI where you can get the text recognition operation result.
    operationLocation = response.getheader('Operation-Location')
    print "Operation-Location:", operationLocation

    parsedLocation = operationLocation.split(serviceUrl)
    answerURL = parsedLocation[1]
    print "AnswerURL:", answerURL

    # 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.

    time.sleep(10)
    conn = httplib.HTTPSConnection(serviceUrl)
    conn.request("GET", answerURL, '', headers)
    response = conn.getresponse()
    print response.status, response.reason
    print response.read()
except Exception as e:
    print e

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, requests, time

requestHeaders = {
    # Request headers - replace this example key with your valid subscription key.
    # Another valid content type is "application/octet-stream".
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

# Replace the three dots below with the URL of a JPEG image containing text.
body = {"url": "..."} 

serviceUrl = 'https://westus.api.cognitive.microsoft.com/vision/v1.0/RecognizeText'

# For printed text. set "handwriting" to false.
```

```
    # Set parameters for the POST request, see documentation for details.
    params = {'handwriting' : 'true'}
```

```
try:
    response = requests.request('post', serviceUrl, json=body, data=None, headers=requestHeaders,
params=params)
    print(response.status_code)

    # This is the URI where you can get the text recognition operation result.
    operationLocation = response.headers['Operation-Location']

    # 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.

    time.sleep(10)
    response = requests.request('get', operationLocation, json=None, data=None, headers=requestHeaders,
params=None)
    data = response.json()
    print(data)
except Exception as e:
    print(e)

#####
```

# Computer Vision Ruby Quick Starts

4/18/2017 • 3 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with Ruby to accomplish the following tasks:

- [Analyze an image](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using Ruby

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:

- The category defined in this [taxonomy](#).
- 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 (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an Image Ruby Example Request

```
require 'net/http'

uri = URI('https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze')
uri.query = URI.encode_www_form({
    # Request parameters
    'visualFeatures' => 'Categories',
    'details' => '{string}',
    'language' => 'en'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# Request headers
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Request body
request.body = "{body}"

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

puts response.body
```

### Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "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
}
}
}

```

## Get a Thumbnail with Computer Vision API Using Ruby

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail Ruby Example Request

```

require 'net/http'

uri = URI('https://westus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail')
uri.query = URI.encode_www_form({
  # Request parameters
  'width' => '{number}',
  'height' => '{number}',
  'smartCropping' => 'true'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# Request headers
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Request body
request.body = "{body}"

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

puts response.body

```

### Get a Thumbnail Response

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

# Optical Character Recognition (OCR) with Computer Vision API Using Ruby

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-readable character stream.

## OCR Ruby Example Request

```
require 'net/http'

uri = URI('https://westus.api.cognitive.microsoft.com/vision/v1.0/ocr')
uri.query = URI.encode_www_form({
    # Request parameters
    'language' => 'unk',
    'detectOrientation' => 'true'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# Request headers
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Request body
request.body = "{body}"

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

puts response.body
```

## OCR Example Response

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

```
{  
    "language": "en",  
    "textAngle": -2.000000000000338,  
    "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"  
                        }  
                    ]  
                },  
                {  
                    "boundingBox": "801,564,153,73",  
                    "text": "IS"  
                }  
            ]  
        }  
    ]  
}
```

# Computer Vision API C# Tutorial

4/12/2017 • 4 min 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.

## 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

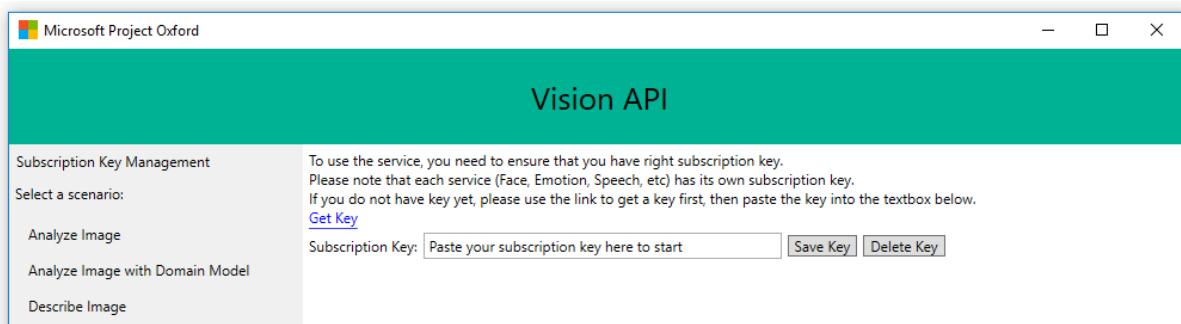
Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services (formerly Project Oxford). For subscription and key management details, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial.

## 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.

Microsoft Project Oxford

## Vision API

Subscription Key Management

Select a scenario:

- Analyze Image
- Analyze Image with Domain Model
- Describe Image
- Generate Tags
- Recognize Text (OCR)
- Get Thumbnail

Analyze an Image

Please click either [Load Image] or paste in an image url and click [Analyze]

Analyzing Done



```
[21:52:43.386402]: Description :  
[21:52:43.386402]: Caption : a man swimming in a pool of water; Confidence : 0.752564820236237  
[21:52:43.386402]: Tags : water, person, sport, swimming, pool,  
[21:52:43.402029]: Tags :  
[21:52:43.402029]: Name : water; Confidence : 0.999414682388306; Hint :  
[21:52:43.402029]: Name : person; Confidence : 0.936775147914886; Hint :  
[21:52:43.417652]: Name : sport; Confidence : 0.848687767982483; Hint :  
[21:52:43.417652]: Name : swimming; Confidence : 0.845447421073914; Hint : sport  
[21:52:43.433278]: Name : water sport; Confidence : 0.827535569667816; Hint : sport  
[21:52:43.433278]: Name : pool; Confidence : 0.805495202541351; Hint :
```

Microsoft will receive the images you upload and may use them to improve Project Oxford and related services. By submitting an image, you confirm you have consent f

4. 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(imagePath))
    {
        //
        // 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 Python Tutorial

4/12/2017 • 1 min 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://westus.api.cognitive.microsoft.com/vision/v1/analyses'  
_key = None #Here you have to paste your primary key  
_maxNumRetries = 10
```

# 86-Categories Taxonomy

4/12/2017 • 1 min 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

# Computer Vision API Frequently Asked Questions

4/18/2017 • 2 min 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-premise 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.

---

# The Research Behind Computer Vision API

4/12/2017 • 1 min to read • [Edit Online](#)

Hao Fang, Saurabh Gupta, Forrest Landola, Rupesh Srivastava, Li Deng, Piotr Dollar, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John Platt, Lawrence Zitnick, and Geoffrey Zweig, *From Captions to Visual Concepts and Back*, CVPR, June 2015 (**Won 1st Prize at the COCO Image Captioning Challenge 2015**)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. arXiv (**Won both ImageNet and MS COCO competitions 2015**)

Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, Jianfeng Gao, *MS-Celeb-1M: Challenge of Recognizing One Million Celebrities in the Real World*, IS&T International Symposium on Electronic Imaging, 2016

Xiao Zhang, Lei Zhang, Xin-Jing Wang, Heung-Yeung Shum, *Finding Celebrities in Billions of Web Images*, accepted by IEEE Transaction on Multimedia, 2012

# Overview

4/12/2017 • 1 min to read • [Edit Online](#)

Content moderation is the process of monitoring User Generated Content (UGC) on online and social media web sites, chat and messaging platforms, and peer communication platforms to track, flag, assess and filter out offensive and unwanted content that creates risks for businesses. The content can include text, images, and videos.

## Where would you use it

Content moderation for all three types of content has several benefits. They are

- Image moderation works great for profile pictures, social media, business documents, and image sharing sites.
- Text moderation benefits communities, family-based web sites, in-game communities, chat and messaging platforms, and user-generated content marketing.
- Video moderation is used for video publishing sites, news sites, and video content sites, to take a few examples.

## Three ways to moderate content

- Automated moderation applies machine learning and AI to cost-effectively moderate at scale
- Human moderation uses teams and the community to moderate all content.
- Hybrid moderation combines automated moderation augmented by the human-in-the-loop (human oversight).

Microsoft Content Moderator enables all three scenarios.

## Get started with the human review tool

The online [human review tool](#) makes it easy to try the automated moderation APIs. It helps augment automated moderation with human-in-the-loop capabilities. It internally calls the automated moderation APIs and make the reviews available right within your web browser. You can invite other users, track pending invites, and assign permissions to your team members.

Use the [review API](#) to auto-moderate content in bulk and review the tagged images or text within the review tool. Provide your API callback point so that you get notified when the reviewers submit their decisions. This allows you to automate the post-review workflow by integrating with your own systems.

## Directly use the automated moderation APIs

You can sign up for the free tiers of [text moderation](#) and [image moderation](#) APIs and apply to use the private preview of the [video moderation](#) APIs to automatically moderate large amount of content and integrate with your review tools and processes.

# Get started with Content Moderator

4/20/2017 • 2 min to read • [Edit Online](#)

## 1. Create a review team

Sign up to try the [human review tool](#). Then upload images or submit sample text to try the automated moderation and human review capabilities without writing any code.

Also read: [Review Tool User Guide](#)

### a. Sign up and invite others

Sign up to try the [review tool](#) by either using your existing Microsoft account or create an account within the review tool. Optionally, invite your colleagues by entering their email addresses.

Let's build your team, they will get an email to sign up and login

Team Name :  i

Email(s) :

Create Team

### b. Upload images or enter text

Use the File Upload feature to upload a set of sample images or enter your text for moderation.

File Upload   Review   Result   Settings

UPLOAD FILES TO REVIEW

Choose your own files or download [sample content](#).

Choose Files

File must be 2 mb or less.

### c. Submit for automated moderation

Submit your content for automated moderation. Internally, the review tool calls the moderation APIs to scan your content. Once the scanning is complete, you see a message informing you about the results waiting for your review.

Name	File Size	File Type	
	sample6.png	618685	image/png
	sample5.png	595913	image/png
	sample2.jpg	273405	image/jpeg
	sample4.png	698749	image/png
	sample1.jpg	17280	image/jpeg
<b>Submit</b>			

#### d. Review and confirm results

As your business application calls the Moderator APIs, the tagged content starts queuing up, ready to be reviewed by the human review teams. You can quickly review large volumes of content using this approach. You are doing a few different things as part of your moderation workflow such as browsing the tagged content, changing the tags, and submitting your decisions.

The screenshot shows a 'Reviews' interface with the following details:

- Reviews to display: 12
- tagged (5) | untagged (3)
- Grid of 7 images:
  - Image 1: Adult (a), Racy (r), Clear
  - Image 2: Adult (a), Racy (r), Clear
  - Image 3: Adult (a), Racy (r), Clear
  - Image 4: Adult (a), Racy (r), Clear
  - Image 5: Adult (a), Racy (r), Clear
  - Image 6: Adult (a), Racy (r), Clear
  - Image 7: Adult (a), Racy (r), Clear
- Next >>

## 2. Directly call the moderation APIs ("scan")

Use your Content Moderator free tier keys available in the **Credentials** TAB under **Settings** to directly try the image and text moderation APIs. You can use the Image APIs to scan images from URLs or binary data. You can use the Text APIs to scan up to a maximum of 1024 characters at a time for profanity, adult, racy, and offensive content. You can either use the "**Try API**" test console within the API reference or write your own application. When you are ready to purchase, you can [upgrade to a paid subscription](#) and swap out the keys in your application.

The screenshot shows the Microsoft Azure Content Moderator interface. The top navigation bar includes 'Content Moderator', 'Dashboard', 'Try', 'Review', and a settings gear icon. Below the navigation is a secondary menu with options: 'Team', 'Tags', 'Connectors', 'Workflows', and 'Credentials'. The 'Credentials' option is highlighted with a teal background and a green arrow points to it from the top navigation. The main content area is titled 'Credentials' and contains the following fields:

- Endpoint : <https://westus.api.cognitive.microsoft.com/contentmoderator/review/v1.0>
- Reference : [API Reference](#)
- Ocp-Apim-Subscription-Key: \* (redacted)
- Resource Id: \* (redacted)
- Image List Ids: (redacted)
- Term List Ids: (redacted)
- Base Url: <https://westus.api.cognitive.microsoft.com/contentmoderator/moderate/v1.0>

An 'Edit' button is located at the bottom right of the form.

### 3. Call the review API for best-of-both ("scan and review")

Use your Content Moderator free tier keys as shown in the previous section, to try the review API's **Job** operation. The **Job** operation calls the underlying moderation APIs (Image or Text). Based on the criteria defined in your custom workflow, it creates **reviews** within the review tool. Once the human reviewers have examined the auto-assigned tags and conveyed their agreement of disagreement by changing them, the review API sends all data to your API callback endpoint.

You can either use the "**Try API**" test console within the API reference or write your own application. When you are ready to purchase, you can [upgrade to a paid subscription](#) and swap out the keys in your application.

# Human-in-the-Loop

4/12/2017 • 1 min to read • [Edit Online](#)

Why do we need a human review tool with automated content moderation powered by machine learning and AI?

We think you get the best results when humans and machines work together on harder cases. Humans can effectively augment machine learning models in situations where the prediction confidence has to be assisted or tempered within a real world context.

Humans can focus on the edge cases and help the models learn and get better over time. The result is a hybrid, automated content moderation process that performs better than if the humans or machines were working alone.

## How the Review tool helps

The human review tool when used in conjunction with our automated moderation APIs allows you to accomplish these important tasks in relation to the content moderation life cycle.

1. Automate the creation of human reviews from the underlying moderation API results
2. Use one tool (Review Tool and API) to moderate multiple formats (text, image, and video - coming soon)
3. Assign or escalate content reviews to multiple review teams organized by content category or experience level.
4. Use default workflows or define custom workflows with flexible rules, and without writing any code.
5. Add human review to any API or business process by simply building a connector.
6. Use the default connectors to review results from Microsoft PhotoDNA, Text Analytics, and Face APIs.
7. Get key performance metrics on your content moderation processes.

# Sign Up for Content Moderator

4/20/2017 • 1 min to read • [Edit Online](#)

Navigate to the [Content Moderator sign up](#) page. You can either sign up with your Microsoft account or create an account on the Content Moderator web site.

The screenshot shows the Microsoft Content Moderator sign-up page. At the top left is the Microsoft logo. In the center, there's a large 'Content Moderator' heading with a subtext: 'Test drive Content Moderator to learn how we enable a complete, configurable content moderation lifecycle by combining automated moderation with human reviews and workflows.' Below this is a 'Sign Up' button. To the right is a 'Sign in' link. The main area features three sections: 'Moderate' (represented by a card icon with '> \_' and the text 'Automated results reduce time and costs'), 'Collaborate' (represented by a network icon with a checkmark and the text 'Manage teams for increased productivity'), and 'Review' (represented by a dollar sign icon and the text 'Approve or reject content for better filtering').

## Name Your Team

Provide a name for your review team. Optionally, you can invite your colleagues to join your team.

The screenshot shows a form for naming a review team. It includes a 'Team Name : \*' field with a note 'Team Name can only use : a-z A-Z 0-9' and an information icon. Below it is an 'Invite others :' field with a note 'Enter email addresses separated by commas.' At the bottom is a dark green 'Create Team' button.

Before you begin, let's create a team to help you collaborate on administering and using the review tool. Optionally, invite others now to join your team.

Team Name : \*

Team Name can only use : a-z A-Z 0-9

Invite others :

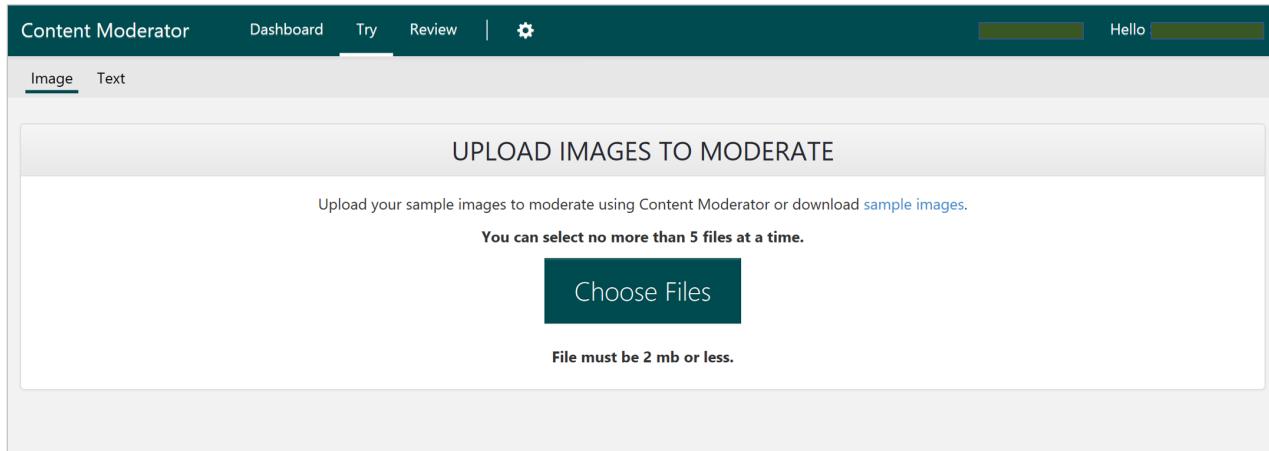
Enter email addresses separated by commas.

Create Team

# Upload Images

4/20/2017 • 1 min to read • [Edit Online](#)

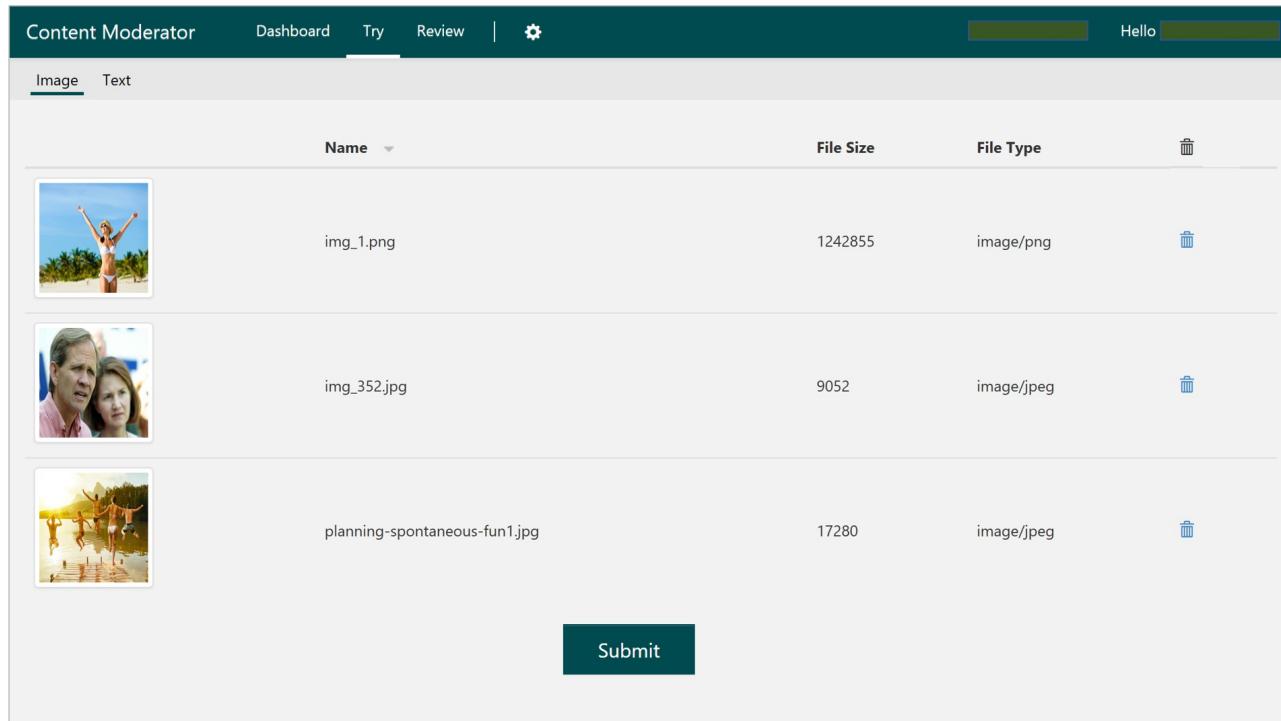
To explore the user interface, upload up to five images at a time, each image not to exceed 2 MB in file size. Be sure to check your file sizes before uploading.



The screenshot shows the 'Content Moderator' interface with the 'Try' tab selected. In the 'Image' section, there is a large button labeled 'Choose Files' for uploading images. A note below it specifies that files must be 2 MB or less. There is also a link to download sample images.

## Submit for Moderation

If you end up uploading more than five images at a time, use the Trashcan icon to remove unwanted images. Then, submit the images for automated moderation.



The screenshot shows the 'Content Moderator' interface with the 'Try' tab selected. Three images have been uploaded and are listed in a table. Each row includes a thumbnail, the image name, file size, file type, and a trashcan icon for deletion. At the bottom, there is a large 'Submit' button.

Name	File Size	File Type	
img_1.png	1242855	image/png	
img_352.jpg	9052	image/jpeg	
planning-spontaneous-fun1.jpg	17280	image/jpeg	

# Review moderated images

4/20/2017 • 1 min to read • [Edit Online](#)

Click the **Review TAB** to browse the auto-moderated images that are ready for review. Any labels assigned to the images are based on the default score thresholds for tagging the images. These thresholds are configurable.

The screenshot shows the Content Moderator interface with the 'Review' tab selected. At the top, there are tabs for 'Image' and 'Text', and a dropdown for 'Default'. Below that, a slider is set to 'Reviews to display: 2'. To the right, there are links for 'tagged (2)' and 'untagged (0)'. The main area displays two images side-by-side. The left image shows a woman in a white bikini on a beach with her arms raised. The right image shows four people jumping off a wooden dock into water. Each image has a 'Clear' button at the bottom right. Below each image, there are two buttons: 'a' (selected) and 'r'. Underneath the images, detailed moderation results are shown:

Id:	adultScore	racyScore	ExternalId
20170108943847f84f4706be...	0.114	0.9	img_1.png
	a False	r True	

On the far right, there is a 'Next >>' button.

Note that the images that you see on your screen are not available to other reviewers in your team while you are reviewing them.

## Zoom in/Zoom out to adjust the number of images

You can move the Reviews to display slider at the top of the screen to adjust the number of images displayed on the screen. Click on the tagged or untagged links to sort the images by tagged or untagged status respectively. Click the tags to toggle their selection status. You can also select any custom tags that you may have created.

## Reviews

Reviews to display: 2

tagged (1) | untagged (1)

Your moderation results are displayed here with the detected tags highlighted. Click the tags to clear or select them. Use the Next button to submit your decisions and review the next set of results.



a r

Id: 2016120e1dbcbb2c1c413ea63685b75567c750

By: [redacted]

Clear

adultScore: 0.114  
racyScore: 0.447  
ExternalId: planning-spontaneous-fun1.jpg



a r

Id: 2016128d2056f2190b4584848399b49567487e

By: [redacted]

Clear

adultScore: 0.009  
racyScore: 0.011  
ExternalId: img\_197.jpg

→ Next >>

## Click a thumbnail to view details

If you have thumbnails showing on your screen, click a thumbnail to bring up the detailed view. Click the tags to toggle their selection status. You can also select any custom tags that you may have created.

2016120e1dbcbb2c1c413ea63685b75567c750

A large thumbnail version of the first image, showing four people jumping off a wooden dock into a lake at sunset. The image is identical to the one in the top section.

Id: 2016120e1dbcbb2c1c413ea63685b75567c750 By: [redacted]

adultScore: 0.114  
racyScore: 0.447  
ExternalId: planning-spontaneous-fun1.jpg

a r

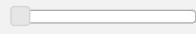
Clear

## Submit image reviews

Once you have reviewed and updated the tag selections, click the **Next** button to submit your reviews.

## Reviews

Reviews to display: 1



tagged (1) | untagged (0)

Your moderation results are displayed here with the detected tags highlighted. Click the tags to clear or select them. Use the Next button to submit your decisions and review the next set of results.



Id: 201612fe69f491e62848189b563efda5c36acb

By: Sanjeev Jagtap

adultScore: 0.114  
racyScore: 0.9  
ExternalId: img\_1.png

a: False  
r: True

a r  
Clear

<< Prev ←

→ Next >>

After you submit, you have about 5 seconds to use the Previous button to go back to your previous screen if you wanted to review them again. After that, the images will be out of the Submit queue and the **Previous** button will disappear from the screen.

# Text Moderation Reviews

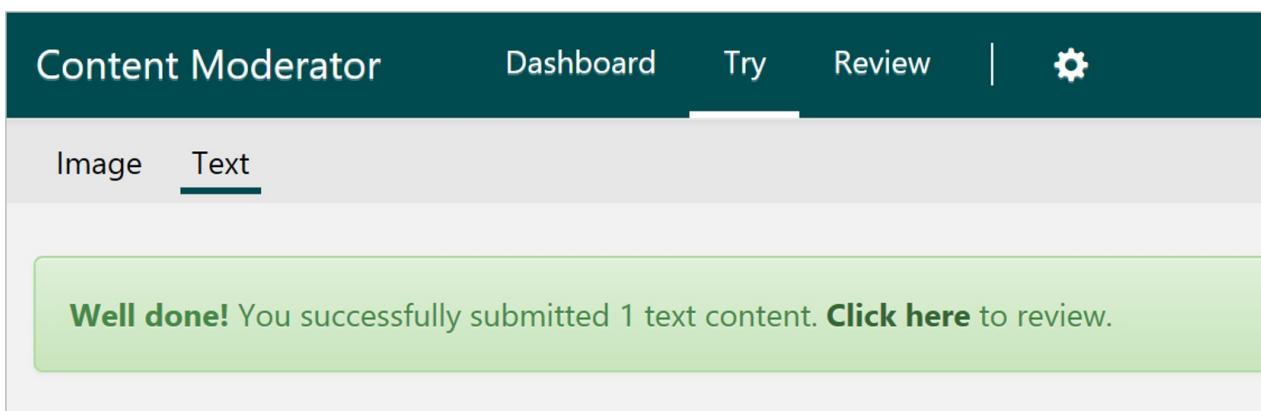
4/21/2017 • 1 min to read • [Edit Online](#)

## Select or enter the text to review

Click the **Try** TAB and select the "Text" option to bring up the Text Moderation start screen. Enter any text upto a maximum of 1024 characters or use the default sample text to submit for automated text moderation.

## Get ready to review results

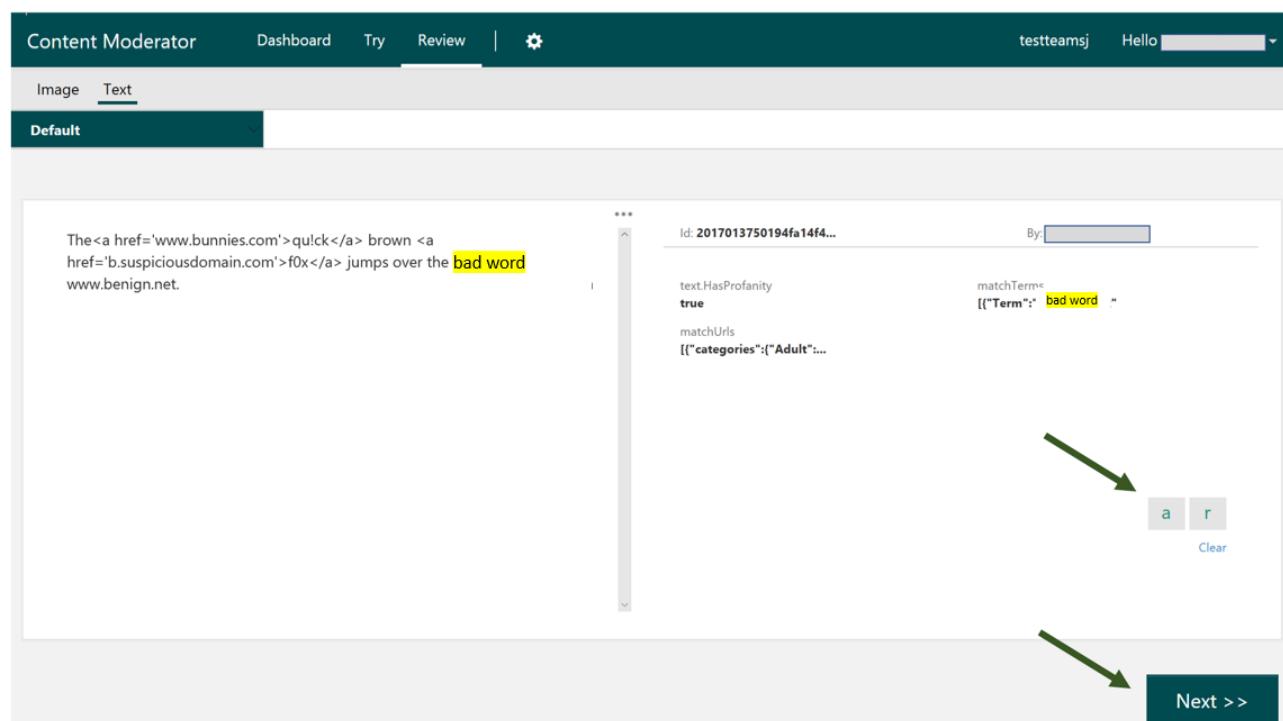
The review tool will call the text moderation API and generate text reviews with the detected tags and match scores results within the review tool for your team's attention.



The screenshot shows the 'Content Moderator' interface with the 'Try' tab selected. Under the 'Text' tab, a green notification box displays the message: 'Well done! You successfully submitted 1 text content. [Click here](#) to review.'

## Review text results

You will see detailed results including the detected tags and terms returned by the text moderation API show up on your screen. Click the tags to toggle their selection status. You can also work with any custom tags that you may have created.



The screenshot shows the 'Content Moderator' interface with the 'Review' tab selected. Under the 'Text' tab, a result for the text 'The <a href='www.bunnies.com'>quck</a> brown <a href='b.suspiciousdomain.com'>f0x</a> jumps over the **bad word** www.benign.net.' is displayed. The result includes:

- Id: 2017013750194fa14f4...
- By: [redacted]
- text.HasProfanity: true
- matchTerms: [{"Term": "bad word"}]
- matchUrls: [{"categories": {"Adult": ...}}]

Below the results, there are two buttons: 'a' and 'r' with a 'Clear' link, and a 'Next >>' button. Two green arrows point from the text above to the 'a' and 'r' buttons.

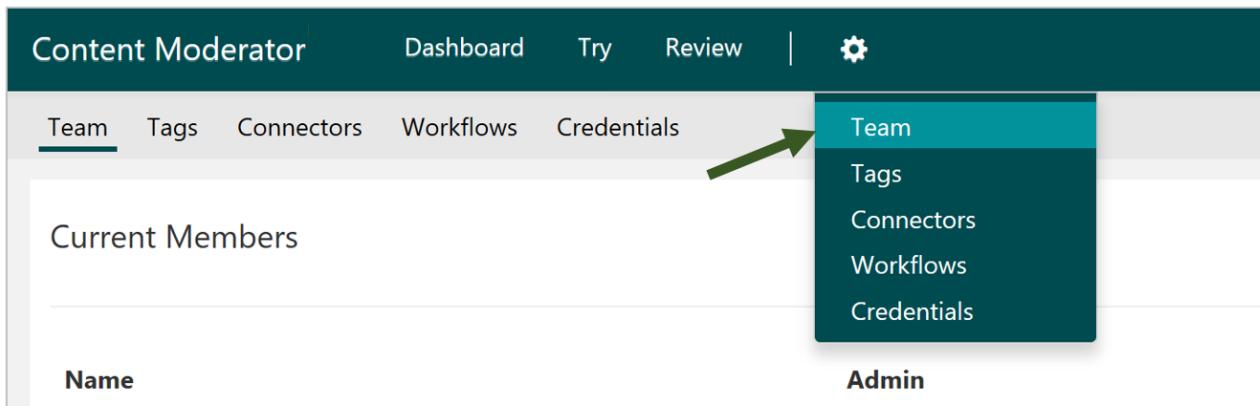
# Team and Subteams

4/20/2017 • 1 min to read • [Edit Online](#)

When you sign up and name your team, that creates your default team. In addition, you can create **subteams** within the review tool. Subteams are useful for creating escalation teams or teams dedicated to reviewing specific categories of content, for example, adult content.

## Go to the Teams Setting

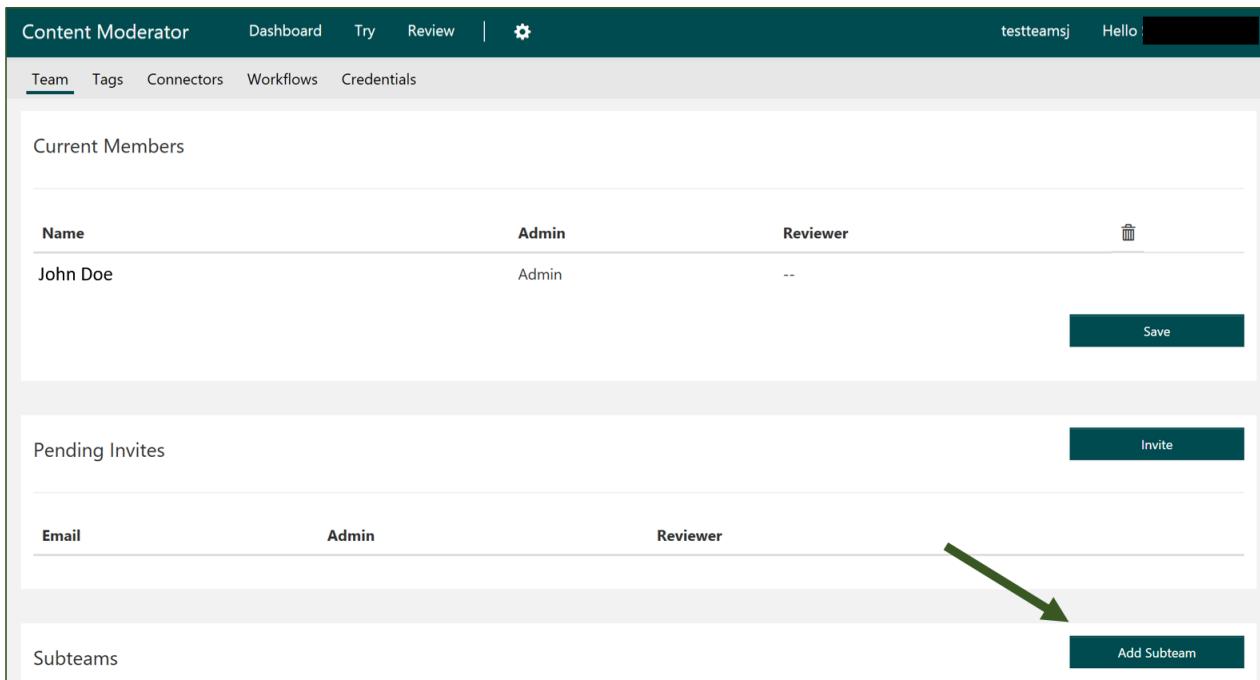
To get started on creating a subteam, select the **Teams** option under Settings.



The screenshot shows the Content Moderator interface. At the top, there's a navigation bar with tabs: Content Moderator, Dashboard, Try, Review, and a settings gear icon. Below the navigation bar, there's a secondary navigation bar with tabs: Team (which is underlined), Tags, Connectors, Workflows, and Credentials. To the right of this secondary bar, there's a dropdown menu with the same five options: Team, Tags, Connectors, Workflows, and Credentials. The 'Team' option in the dropdown is highlighted with a teal background. The main content area has sections for 'Current Members' and 'Pending Invites', each with tables showing names, roles, and actions like 'Save' or 'Invite'. At the bottom, there's a section for 'Subteams' with a 'Add Subteam' button.

## Use the Subteam command

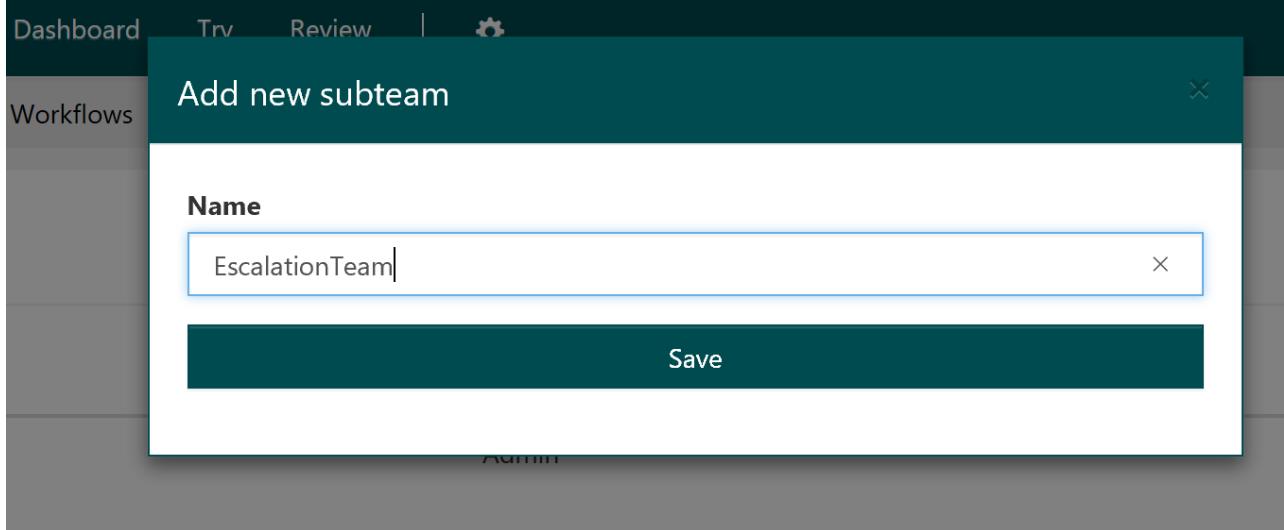
Scroll down the page and use the "Add Subteam" button to bring up the name dialog.



The screenshot shows the Content Moderator interface with the 'Subteams' section visible. There's a large green arrow pointing from the 'Add Subteam' button at the bottom right of the 'Subteams' section towards a modal dialog. The modal dialog has a single input field labeled 'Name' with the placeholder 'Subteam Name' and a 'Save' button below it.

## Name your subteam

Enter your subteam name in the dialog.



## Assign members from your default team

Use the "**Add Member**" option to assign members from your default team to one or more subteams. You can only add existing users to a subteam. For adding new users who are not in the review tool, invite them by using the "Invite" button on the Team Settings page.

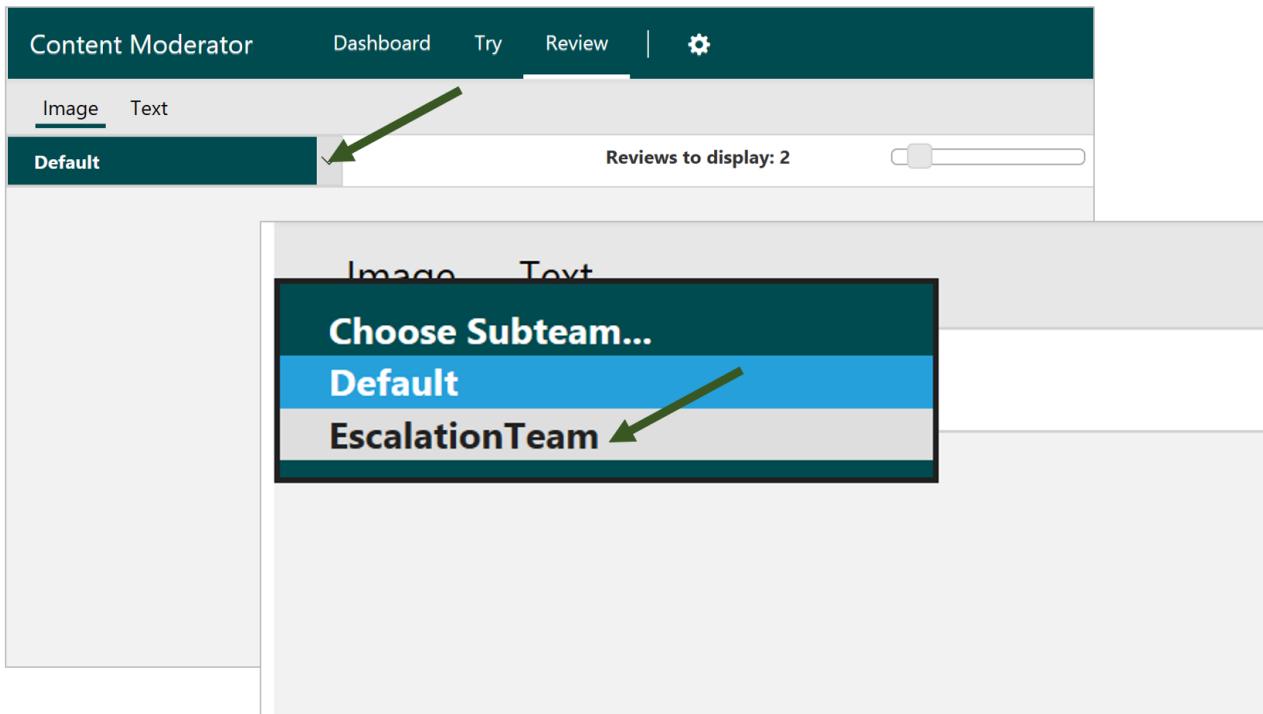
## Assign reviews to your subteam

Once you have your subteams created and team members assigned, you can start assigning image and text reviews to those subteams. The following screen shows from where you can do so during the review process.

For images:

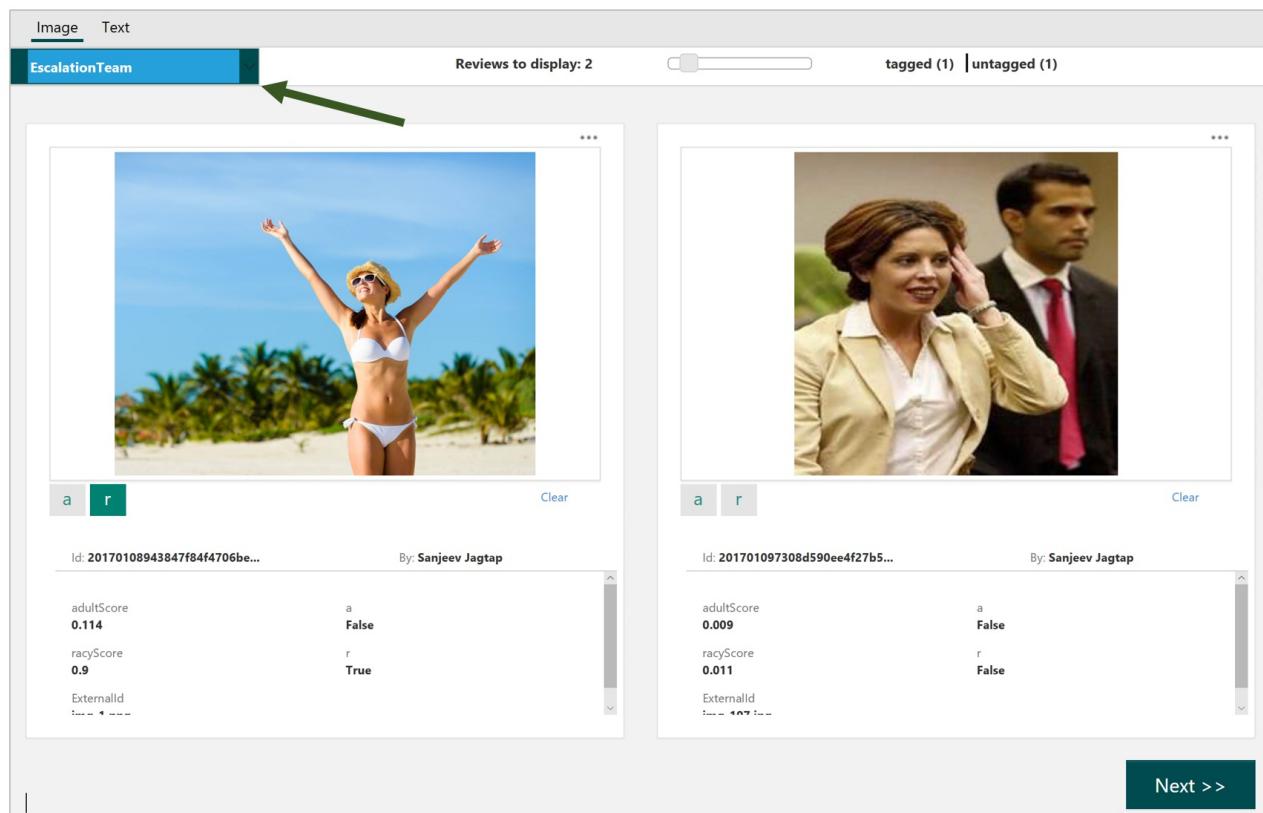
## Switch between subteams to review assigned content

If you are a member of one or more subteams, you can switch between your subteams as shown in the following screenshot.



## Review content within a subteam

Once you have your subteam selected, you will see all assigned and still pending content reviews for your attention.



# Defining and using workflows

4/20/2017 • 1 min to read • [Edit Online](#)

In addition to default workflow used for generating reviews, you can define custom workflows and thresholds based on content policies that are specific to your business. Content Moderator allows you to use other APIs in addition to its own API as long as a connector for that API is available.

## Make sure you have valid credentials

To get started on defining a workflow, make sure you have valid credentials for the API you intend to use in your workflow. Content Moderator includes a small set of Connectors by default.

The screenshot shows the Content Moderator dashboard. In the top navigation bar, 'Connectors' is selected. A dropdown menu also shows 'Connectors' as the active option. Below this, the 'Connected Connectors' section lists a single item: 'Content Moderator' with a description 'Enables using all Content Moderator APIs in your workflows'. In the 'Available Connectors' section, two connectors are listed: 'Face API' and 'PhotoDNA Cloud Service'. The 'PhotoDNA Cloud Service' card is highlighted with a green arrow pointing to it. To the right, a modal window titled 'Add Connector' is open, prompting for 'Name' (set to 'PhotoDNA Cloud Service'), 'Subscription Key\*' (a placeholder 'Your PhotoDNA subscription key'), and 'Host Url\*' (a placeholder 'PhotoDNA Host Url'). A green arrow points from the 'PhotoDNA Cloud Service' card to this modal window.

## Navigate to the Workflows section

Select the **Workflows** option under **Settings**.

The screenshot shows the Content Moderator dashboard with the 'Review' tab selected. A dropdown menu in the top right corner shows the 'Workflows' option highlighted with a green arrow. Other options in the menu include 'Team', 'Tags', 'Connectors', 'Workflows', and 'Credentials'.

## Start a new workflow

Use the **Add Workflows** option to get started.

The screenshot shows the Content Moderator dashboard with the 'Workflows' tab selected. A green arrow points to the 'Add Workflow' button in the top right corner of the main content area. The table below lists one workflow named 'default' with a type of '0'.

Name	Description	Type	Actions
default	Default	0	

## Name your workflow

Name your workflow, provide a description, and select whether you want to process images or text. In the screenshot below, you can see the fields and view the If-Then-Else selections that you need to make to define your custom workflows.

The screenshot shows the 'Add Workflow' dialog. A green arrow points to the 'Name' field containing 'AdultOnlyWorkflow'. Another green arrow points to the 'Content Type' dropdown, which is set to 'Image'. A third green arrow points to the 'If:' section under the 'Designer' tab, which contains a dropdown labeled 'Choose type of condition...'. The 'JSON' tab is also visible at the bottom.

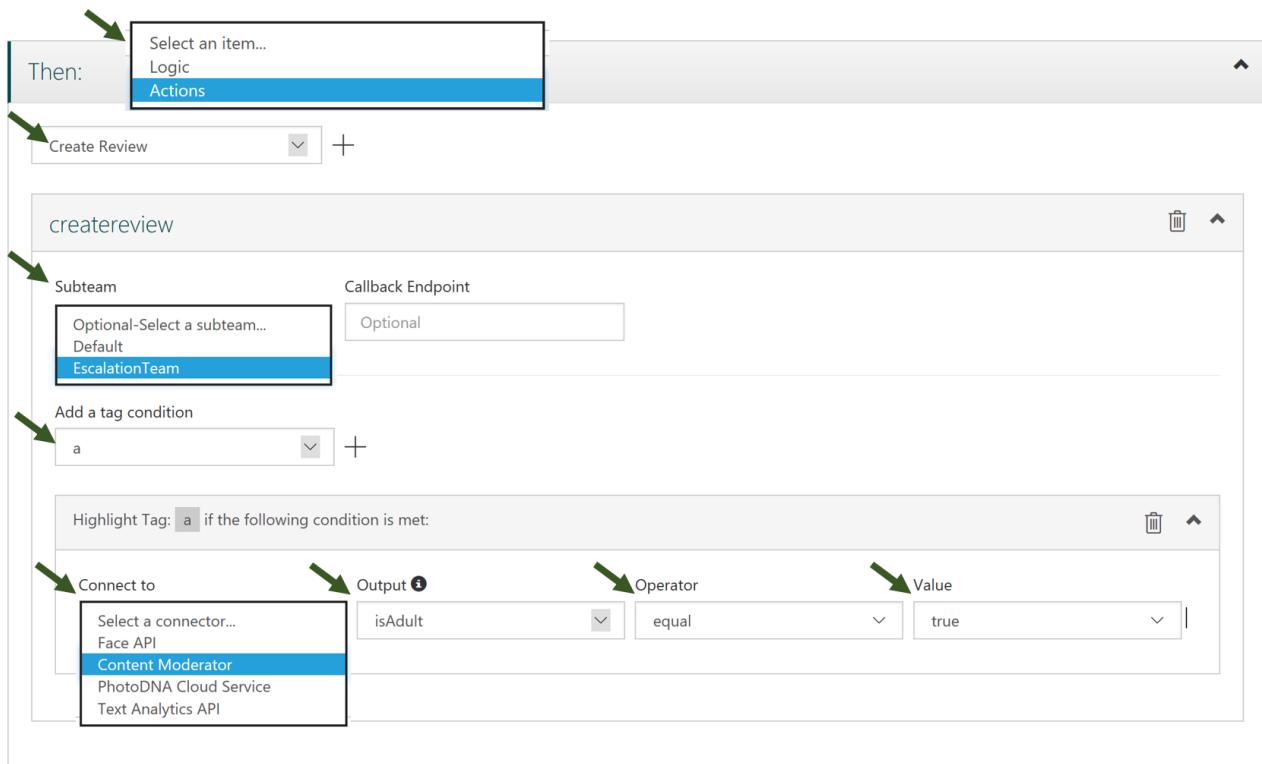
## Define the evaluation criteria (condition)

As a first step, enter all the information needed to define your criteria for executing the workflow. As shown in the screen below, this includes selecting the API you want to get results from. When you select one of the available APIs (that you have entered your credentials for in the very first step), the next drop-down will show the available outputs from the API. The next two fields allow you to specify the check to be performed.

The screenshot shows the 'If:' condition configuration. A green arrow points to the 'Condition' dropdown, which is set to 'isAdult'. Another green arrow points to the 'Connect to' dropdown, which is set to 'Content Moderator'. A third green arrow points to the 'Value' field, which is set to 'true'. The 'Designer' tab is selected at the top.

## Define the action

Once you have defined the condition, you will tell Content Moderator what action to perform if the condition is met. The example shown below creates an image review and assigns it to a subteam. It also specifies an additional criteria that must be fulfilled for the assigned 'a' tag to be selected. In this way, you can combine multiple conditions to get the results you want.



## Optionally, define the Else section

Optionally, expand the **Else** section to provide similar information like you did for the **If** section.

Else:

Actions

Create Review +

createreview

Subteam      Callback Endpoint

Optional-Select a subteam...      Optional

Add a tag condition

Select a tag... +

This screenshot shows the 'Else' block configuration in the Workflow Editor. It includes an 'Actions' dropdown, a 'Create Review' action, a condition named 'createreview', and various configuration fields for subteams and callbacks.

## Save the workflow

Finally, save your workflow and note the workflow name. You will need it to invoke the workflow with the Review API.



## Use the Review API

Now that you have a custom workflow defined, use the [Review API](#) to start a moderation job with the workflow name as one of the parameters. This should be the workflow name that you noted in the previous step.

# Configure or get the Review Tool settings

4/20/2017 • 1 min to read • [Edit Online](#)

## Team

After you have sent out invites, you can monitor them, change permissions for team members, and invite additional users on the screens shown below. You can also create subteams and assign existing members. The users can be either administrators or reviewers. The difference between the two roles is that administrators can invite other users, while reviewers cannot.

The screenshot shows the Content Moderator interface with the 'Team' tab selected. The main area displays 'Current Members' with one entry: Sanjeev Jagtap (Admin). A modal menu is open over this entry, listing 'Team', 'Tags', 'Connectors', 'Workflows', and 'Credentials'. Below the member list is a 'Pending Invites' section with an 'Email' input field and a 'Save' button. At the bottom is a 'Subteams' section containing an 'EscalationTeam' subteam with one member, Sanjeev Jagtap, and buttons for 'Add Subteam' and 'Add Member'.

## Tags

This is where you can define your custom tags by entering the short code, name, and description for your tags and using the **Add** button. You will see your new tag on the screen. Click the **Save** button to save your new tag and make it available during reviews.

The screenshot shows the Content Moderator interface with the 'Tags' tab selected. A context menu is open over the table, with 'Tags' highlighted. In the foreground, a modal window is displayed for adding a new tag. The fields are: Short code: \*mt\*, Name: \*My Tag\*, and Description: \*My custom tag\*. The 'Add' button is at the bottom. After clicking 'Add', the tag is listed in the main table with the ID 'mt'.

	Short code	Name	Help description	Is visible	
	a	isadult	Adult Content	<input checked="" type="checkbox"/>	
	r	isracy	Racy Content	<input checked="" type="checkbox"/>	
	mt	My Tag	My custom tag	<input checked="" type="checkbox"/>	

## Connectors

The Content Moderator review tool calls the Content Moderator APIs with the **default** workflow to moderate content. It will auto-provision the Moderator API credentials for you when you sign up for the review tool.

It also supports integrating other APIs as long as a connector is available. We have made a few connectors available out of the box. You can enter your credentials for these APIs by using the **Connect** button. You can then use these connectors in your custom workflows, as shown in the next section.

Content Moderator    Dashboard    Try    Review    |   

Team    Tags    **Connectors**    Workflows    Credentials

Connected Connectors

Name	Description	Actions
Content Moderator	Enables using all Content Moderator APIs in your workflows	

Available Connectors

**Face API**  
Enables Cognitive Services Face apis in your workflows  
[Learn more](#)

**Connect**

**PhotoDNA Cloud Service**  
Enables PhotoDNA apis in your workflows  
[Learn more](#)

**Connect**

**Text Analytics API**  
Enables Cognitive Services Text Analytics apis in your workflows  
[Learn more](#)

**Connect**

## Workflows

This is where you can see the **default** workflow. You can also define custom workflows for image and text by using the API Connectors available under the Connectors section. For a detailed example of how to define custom workflows, please see the [Workflows](#) section.

Content Moderator    Dashboard    Try    Review    |   

Team    Tags    Connectors    **Workflows**    Credentials

Workflows

Name	Description	Type	Actions
default	Default	Image	

**Add Workflow**

## Credentials

In this section, you can access your Content Moderator subscription key to use with the APIs (Image, Text, List, Workflow, and Review APIs) included with Content Moderator. You will use these APIs to integrate your content with the various Content Moderator capabilities, whether for scanning or for both scanning and enabling human review workflows.

Content Moderator      Dashboard      Try      Review ▾      |     

Team Tags Connectors Workflows Credentials

Credentials

Endpoint : <https://westus.api.cognitive.microsoft.com/contentmoderator/review/v1.0>

Reference : [API Reference](#)

Ocp-Apim-Subscription-Key: \*

Resource Id: \*

Image List Ids:

Term List Ids:

Base Url: <https://westus.api.cognitive.microsoft.com/contentmoderator/moderate/v1.0>

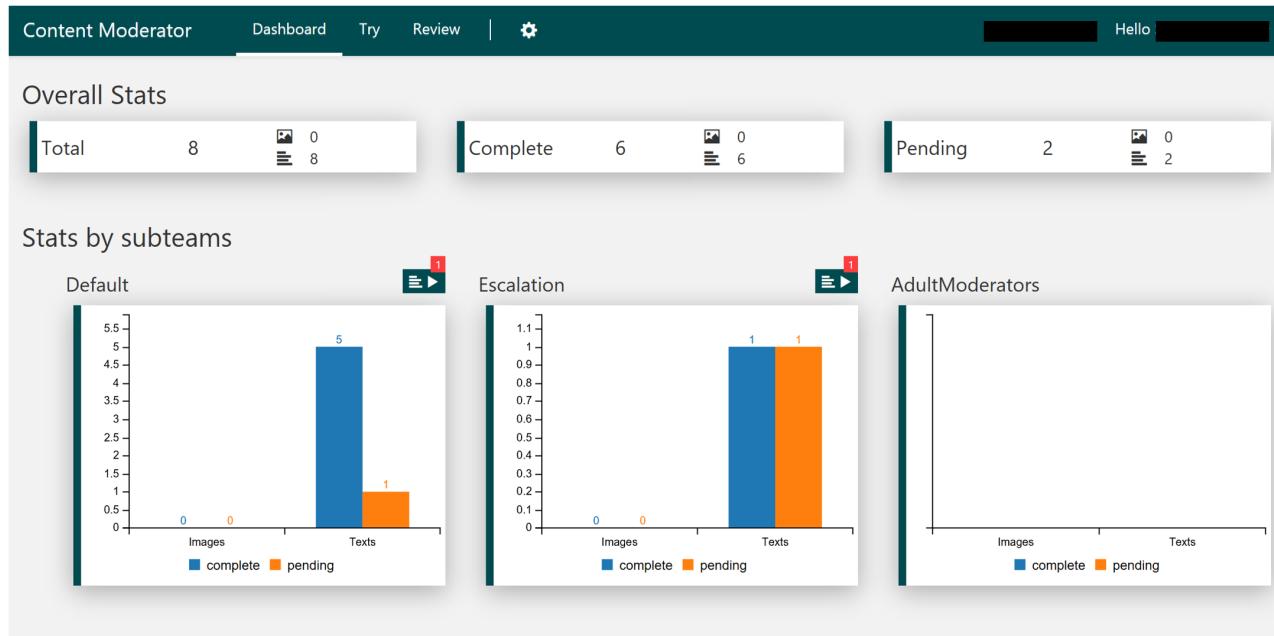
**Edit**

The screenshot shows the Microsoft Content Moderator interface. At the top, there's a dark header with the 'Content Moderator' logo, 'Dashboard', 'Try', 'Review ▾', and a settings gear icon. Below the header is a secondary navigation bar with links: 'Team', 'Tags', 'Connectors', 'Workflows', and 'Credentials'. The 'Credentials' link is underlined and has a green arrow pointing to it from the top-left. The main content area is titled 'Credentials' and contains several input fields: 'Endpoint' (with a value), 'Reference' (with a link), 'Ocp-Apim-Subscription-Key' (marked with an asterisk), 'Resource Id' (marked with an asterisk), 'Image List Ids', 'Term List Ids', and 'Base Url' (with a value). Each input field has an 'Info' icon to its right. At the bottom right of the content area is a large blue 'Edit' button.

# View Dashboard

4/20/2017 • 1 min to read • [Edit Online](#)

Click the **Dashboard TAB** to see key metrics for all content review done within the tool. You can see total, completed, and in-progress reviews for images and text. You can see metrics for your default team and for your subteams.



# API Reference

4/19/2017 • 1 min to read • [Edit Online](#)

You get started with the Content Moderator APIs in the following ways:

1. [Subscribe to the Content Moderator API](#) on the Microsoft Azure portal.
2. Sign up for the [content moderator review tool](#).

If you sign up for the review tool, you will find your free tier key in the **Credentials** TAB under **Settings** as shown in the following screenshot:

The screenshot shows the Microsoft Azure Content Moderator review tool interface. At the top, there's a navigation bar with 'Content Moderator' and tabs for 'Dashboard', 'Try', 'Review', and 'Settings'. A dropdown menu from 'Settings' is open, showing options like 'Team', 'Tags', 'Connectors', 'Workflows', and 'Credentials'. The 'Credentials' tab is selected and highlighted in blue. Below the navigation, there's a section titled 'Credentials' with fields for 'Endpoint' (set to 'https://westus.api.cognitive.microsoft.com/contentmoderator/review/v1.0'), 'Reference' (link to 'API Reference'), and several input fields for 'Ocp-Apim-Subscription-Key', 'Resource Id', 'Image List Ids', 'Term List Ids', and 'Base Url'. Each input field has an info icon (i) to its right. At the bottom right is a dark green 'Edit' button.

## Image Moderation API

Use the image moderation API to scan your images and get back predicted tags, their confidence scores, and other extracted information. Use this information to implement your post-moderation workflow such as publish, reject, or review the content within your systems.

### [Image Moderation API Reference](#)

## Text Moderation API

Use the text moderation API to scan your text content and get back identified profanity terms, predicted tags and confidence scores. Use this information to implement your post-moderation workflow such as publish, reject, or review the content within your systems.

### [Text Moderation API Reference](#)

## Review API

Use the review API to initiate scan-and-review moderation workflows with both image and text content. The

moderation job scans your content by using the image and text moderation APIs. It then uses the default and custom workflows defined within the review tool to generate reviews within the review tool. Once your human moderators have reviewed the auto-assigned tags and prediction data and submitted their final decision, the review API submits all information to your API endpoint.

### [Review API Reference](#)

## Workflow API

Use this API to create, update, and get details of your custom workflows created by your team. You or your team colleagues define these workflows in the review tool. These workflows use Content Moderator or even other APIs available as connectors within the review tool.

### [Workflow API Reference](#)

## List Management API

Use this API to create and manage your custom exclusion or inclusion lists of images and text. If enabled, the **Image/Match** and **Text/Screen** operations do fuzzy matching of the submitted content against your custom lists and skip the ML-based moderation step for efficiency.

### [List Management API Reference](#)

# Image Moderation API

4/19/2017 • 2 min to read • [Edit Online](#)

Use Content Moderator's image moderation API ([see API reference](#)) to moderate images for adult and racy content. Match against custom and shared lists and implement optical character recognition (OCR) and face detection.

## Evaluating for adult and racy content

The Content Moderator API's **Evaluate** operation returns a classification scores (between 0 and 1) and Boolean data (true or false) to indicate whether the image contains adult or racy content. For example, when you call the API with your image (file or URL), you get back information that includes:

- The match score (between 0 and 1)
- A Boolean value (true or false) based on default thresholds and actual score

## Detecting text with Optical Character Recognition (OCR)

The Optical Character Recognition (**OCR**) capability detects text content in an image and extracts it for search and numerous other purposes. If you do not specify a language, the detection defaults to English.

A response includes:

- The original text
- The detected text elements with their confidence scores

## Detecting faces

Detecting faces is important in the context of content moderation because you may not want your users to upload any personally identifiable information (PII) and risk their privacy and your brand. Using the Detect faces operation, you can detect the probability of finding faces and their count in each image.

A response includes:

- Faces count
- List of locations of faces detected

## Matching against your custom list

In many online communities, after users upload images or other type of content, the offensive may get shared multiple times over the following days, weeks, and months. The costs of repeatedly scanning and filtering out the same image or even slightly modified versions of the image from multiple places can be expensive and error-prone.

Instead of moderating the same image multiple times, you may want to add the offensive images to your custom list of blocked content. That way, your content moderation system can compare incoming images against your custom list of images and stop any further processing right away.

The **Match** operation allows fuzzy matching of incoming images against any of your custom lists, created and managed using the List operations.

If found, the operation returns the identifier and the moderation tags of the matched image. The information includes:

- Match score (between 0 & 1)
- Matched image
- Image tags (assigned during a previous moderation)
- Image labels

## Creating and managing your custom lists

As mentioned previously, if the images are already tagged, you would match repeated content against pre-approved or pre-rejected images, without having to go through the moderation-and-review workflow.

The Content Moderator provides a complete list management API with operations for creating and deleting lists, and for adding and removing images from those lists.

A typical sequence of operations would be to:

1. Create a list.
2. Add images to your list.
3. Match images against the ones in the list.
4. Delete image from the list.
5. Delete the list.

# Text Moderation API

4/19/2017 • 1 min to read • [Edit Online](#)

Use Content Moderator's text moderation API ([see API reference](#)) to moderate text for profanity in more than 100 languages, and match against custom and shared lists that are specific to your business and users.

## Language detection

The first step to using the text moderation API is to have the algorithm detect the language of the content to be moderated. The API supports more than [100 languages](#). The **Detect Language** operation returns language codes for the predominant language comprising the submitted text in the following format: {"DetectedLanguage": "eng"}

## Screening for profanity

The text moderation API's **Screen** operation does it all – screen the incoming text (maximum 1024 characters) for profanity and PII, while matching against custom lists of terms.

The response may include:

- Auto-corrected text
- Original text
- Language
- PII
- Location of detected terms within the submitted text
- Terms: detected profanity content

Let's look at these fields in greater detail.

## Auto-correction

Let's assume that the input text is: (the 'Izay' is intentional.)

```
The <a href="www.bunnies.com">qu!ck</a> brown <a href="b.suspiciousdomain.com">f0x</a> jumps over the lzay dog  
www.benign.net.
```

If you ask for auto-correction, the response will contain the corrected version of the text as in:

```
"The quick brown fox jumps over the lazy dog."
```

## Profanity terms

If any terms are detected, those terms are included in the response, along with their starting index (location) within the original text.

## Creating and managing your custom lists of terms

While the default, global list of terms works great for most cases, you may want to screen against terms that are specific to your business needs. For example, you may want to filter out any competitive brand names from posts by users. Your threshold of permitted text content may be different from the default list.

The Content Moderator provides a complete [terms list API](#) with operations for creating and deleting lists of terms, and for adding and removing text terms from those lists.

A typical sequence of operations would be to:

1. Create a list.
2. Add terms to your list.
3. Screen terms against the ones in the list.
4. Delete term or terms from the list.
5. Delete the list.

# Text Moderation API - Supported Languages

4/12/2017 • 1 min to read • [Edit Online](#)

LANGUAGE	CODE
Afrikaans	afr
Albanian	sqi
Amharic	amh
Arabic	ara
Armenian	hye
Assamese	asm
Azerbaijani	aze
Bangla Bangladesh	bng
Bangla India	ben
Basque	eus
Belarusian	bel
Bosnian Cyrillic	bos
Bosnian Latin	bsb
Breton [non-GeoPol]	bre
Bulgarian	bul
Catalan	cat
Central Kurdish	kur
Cherokee	bul
Catalan	chr
Chinese (Simplified)	zho
Chinese (Traditional) Hong Kong SAR	zh-hk
Chinese (Traditional) Taiwan	zh-tw

LANGUAGE	CODE
Croatian	hrv
Czech	ces
Danish	dan
Dari	prs
Dutch	nld
English	eng
Estonian	est
Filipino	fil
Finnish	fin
French	fra
Galician	glg
Georgian	kat
German	deu
Greek	ell
Gujarati	guj
Hausa	hau
Hebrew	heb
Hindi	hin
Hungarian	hun
Icelandic	isl
Igbo	ibo
Indonesian	ind
Inuktitut	iku
Irish	gle
isiXhosa	xho

LANGUAGE	CODE
isiZulu	zul
Italian	ita
Japanese	jpn
Kannada	kan
Kazakh	kaz
Khmer	khm
K'iche	quc
Kinyarwanda	kin
Kiswahili	swa
Konkani	kok
Korean	kor
Kyrgyz	kir
Lao	lao
Latvian	lav
Lithuanian	lit
Luxembourgish	ltz
Macedonian	mac
Malay	msa
Malayalam	mym
Maltese	mlt
Maori	mri
Marathi	mar
Mongolian	mon
Nepali	nep
Norwegian (Bokmål)	nob

LANGUAGE	CODE
Norwegian (Nynorsk)	nno
Odia	ori
Pashto	pus
Persian	fas
Polish	pol
Portuguese Brazil	por
Portuguese Portugal	1or
Pulaar	ful
Punjabi	pan
Punjabi (Pakistan)	pnb
Quechua (Peru)	qup
Romanian	ron
Russian	rus
Scottish Gaelic	gla
Serbian (Cyrillic)	srp
Serbian (Cyrillic, Bosnia and Herzegovina)	srn
Serbian (Latin)	1rp
Sesotho	sot
Sesotho sa Leboa	nso
Setswana	tsn
Sinhala	sin
Slovak	slk
Slovenian	slv
Spanish	spa
Swedish	swe

LANGUAGE	CODE
Tajik	tgk
Tamil	tam
Tatar	tat
Telugu	tel
Thai	tha
Tigrinya	tir
Turkish	tur
Turkmen	tuk
Ukrainian	ukr
Urdu	urd
Uyghur	uig
Uzbek	uzb
Valencian	cat
Vietnamese	vie
Welsh	cym
Wolof	wol
Yoruba	yor

# The Review API

4/19/2017 • 3 min to read • [Edit Online](#)

Use Content Moderators's review API ([See API reference](#)) to integrate your content with the review tool to get both automated moderation and human reviews.

The Review API has a small set of operations that use the underlying moderation APIs to moderate your content and make the tagged images available within the review tool for human review.

Here is a sample sequence of steps you might follow:

1. Submit an image to start a moderation job.
2. Get details for a moderation job that is either in-progress or completed.
3. Once the human reviews are completed, get the detailed results.

## Submitting an image to start a moderation job

Use the **Job-Create** operation to start a moderation job for an image. The Moderator will scan the image and generate reviews by evaluating the default or custom workflow configured within the review tool.

Your inputs will include:

- The image to be moderated (file or URL)
- The default workflow identifier ("default")
- Your API callback point for notification

The response will return the identifier of the job that was started.

Once the job is completed, the operation will use your API callback information to notify you. You can then use the job identifier to get the detailed information back and take further action.

## Getting details for an in-progress or completed job

Use the **Job-Get** operation to get the details of a moderation job for the identifier returned by the previous operation. Note that while the method returns immediately with the job settings information, the moderation job itself is run asynchronously. The results are returned through the callback point, and can also be queried by using the operation that we discuss in the next section.

Your inputs will include at a minimum:

- The human review team name
- The job identifier returned by the previous operation

The response will include the following information:

- The identifier of the review created (use this to get the final review results)
- The status of the review (completed or in-progress)
- The assigned moderation tags (key-value pairs)

## Getting results after a human review

Use the **Review-Get** operation to get the results after a human review of the moderated image is completed. You will get notified when your defined callback point is called by the review API. The operation will return two sets of

tag data - the tags assigned by the moderation service, and the tags after the human review was completed.

Your inputs will include at a minimum:

- The human review team name
- The review identifier returned by the previous operation

The response will include the following information:

- The tags (key-value pairs) confirmed by the human reviewer
- The tags (key-value pairs) assigned by the moderation service

## Directly creating reviews within the review tool

If you want to use the review tool just for the human review capability because you are either already using the moderation APIs, or you want to separate them for other reasons, you can do that using the **Review-Create** operation.

Your inputs to this operation will include:

- The content (image) to be reviewed
- The tags (key value pairs)

## Creating or updating custom workflow

While the review tool uses a default workflow, you can also define your custom workflow based on your business rules and content policies.

You would do that using the **Workflow-Create or Update** operation.

Your inputs to this operation will include:

- The review team name
- The name of your workflow

Your request will contain the expression for describing the workflow. When creating Jobs, the Expression in the specified workflow is evaluated against the moderator API results.

For example:

Expression with one condition as in: Create review if **adult score > 0.4**.

```
{  
  "ConnectorName": "imagemoderator",  
  "OutputName": "adultscore",  
  "Operator": "ge",  // where ge = greater than or equal to  
  "Value": "0.4",  
  "Type": "Condition"  
}
```

Expression with two conditions combined as in: Create review if **(adult score > 0.5 AND racy score > 0.5)**.

```
{  
    "Left": {  
        "ConnectorName": "imagemoderator",  
        "OutputName": "adultscore",  
        "Operator": "ge",  
        "Value": "0.4",  
        "Type": "Condition"  
    },  
    "Right": {  
        "ConnectorName": "imagemoderator",  
        "OutputName": "racyscore",  
        "Operator": "ge",  
        "Value": "0.5",  
        "Type": "Condition"  
    },  
    "Combine": "AND",  
    "Type": "Combine"  
}
```

Please refer to the API Reference for more information and examples.

# Video Moderation API

4/12/2017 • 3 min to read • [Edit Online](#)

Today, online viewers are generating billions of video views across popular and regional social media web sites and increasing. With proactive detection of adult content in your video files, you can significantly lower the cost of your moderation efforts. Furthermore, gaining this type of early insight can enable you to make better decisions and create safer experiences for your customers.

## How it works

The Content Moderator video moderation capability is powered by the adult content classifier running within the **Azure Media Services**. The capability is currently in private preview and available at no charge. Here are the steps you need to follow to try the service:

1. [Sign up](#) for a free Microsoft Azure subscription if you don't have one already.
2. [Submit a support request](#) for enabling the East US 2 region for your Azure Subscription ID.
3. [Contact us](#) with your Azure Subscription ID for enabling the Video Content Moderator in the East US 2 region.
4. [Create a Media Services account](#) in the East US 2 region on the Azure portal.

The free tiers can help you in evaluating the video moderation with your content. Depending on your volumes and purchase options, your expenses will include just the media storage and compute costs.

## Code Sample

The following is a sample C# program set that will get you started with your first Content Moderator job. This code requires both the [Azure Media Services C# SDK](#) and [SDK Extensions packages](#) (available on [NuGet](#)).

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.WindowsAzure.MediaServices.Client;
using System.IO;
using System.Threading;

namespace ContentModeratorAmsSample
{

    class Program
    {
        // declare constants and globals
        private static CloudMediaContext _context = null;
        private static readonly string _accountName = { ACCOUNT_NAME };
        private static readonly string _accountKey = { ACCOUNT_KEY };
        private const string _mpName = "Azure Media Content Moderator";
        private static readonly string _inputFile = { INPUT_FILE_PATH };
        private static readonly string _outputFolder = { OUTPUT_FOLDER_PATH };

        //a json file with the configuration and version the supported Modes are Speed, Balance, Quality, which
        provide Moderator readings over a 16-/32-/48-frame granularity.
        //Example file:
        //      {
        //          "version": "1.0",
        //          "Options": { "Mode": "Quality" }
        //}

        private static readonly string _moderatorConfiguration = { CONFIGURATION_FILE_PATH };
```

```

static void Main(string[] args)
{
    _context = new CloudMediaContext(_accountName, _accountKey);
    string configuration = File.ReadAllText(_moderatorConfiguration);
    RunContentModeratorJob(_inputFile, _outputFolder, configuration);
}

static void RunContentModeratorJob(string inputFilePath, string output, string configuration)
{
    // create asset with input file
    IAsset asset = _context.Assets.CreateFromFile(inputFilePath, AssetCreationOptions.None);

    // grab instance of Azure Media Content Moderator MP
    IMediaProcessor mp = _context.MediaProcessors.GetLatestMediaProcessorByName(_mpName);

    // create Job with Content Moderator task
    IJob job = _context.Jobs.Create(String.Format("Content Moderator {0}",
        Path.GetFileName(inputFilePath) + "_" + Guid.NewGuid()));

    ITask contentModeratorTask = job.Tasks.AddNew("Adult classifier task",
        mp, configuration,
        TaskOptions.None);
    contentModeratorTask.InputAssets.Add(asset);
    contentModeratorTask.OutputAssets.AddNew("Adult classifier output",
        AssetCreationOptions.None);

    job.Submit();

    // Create progress printing and querying tasks
    Task progressPrintTask = new Task(() =>
    {
        IJob jobQuery = null;
        do
        {
            var progressContext = _context;
            jobQuery = progressContext.Jobs
                .Where(j => j.Id == job.Id)
                .First();
            Console.WriteLine(string.Format("{0}\t{1}",
                DateTime.Now,
                jobQuery.State));
            Thread.Sleep(10000);
        }
        while (jobQuery.State != JobState.Finished &&
            jobQuery.State != JobState.Error &&
            jobQuery.State != JobState.Canceled);
    });
    progressPrintTask.Start();

    Task progressJobTask = job.GetExecutionProgressTask(
        CancellationToken.None);
    progressJobTask.Wait();

    // If job state is Error, the event handling
    // method for job progress should log errors. Here we check
    // for error state and exit if needed.
    if (job.State == JobState.Error)
    {
        ErrorDetail error = job.Tasks.First().ErrorDetails.First();
        Console.WriteLine(string.Format("Error: {0}. {1}",
            error.Code,
            error.Message));
    }

    DownloadAsset(job.OutputMediaAssets.First(), output);
}

static void DownloadAsset(IAsset asset, string outputDirectory)

```

```

    {
        foreach (IAssetFile file in asset.AssetFiles)
        {
            file.Download(Path.Combine(outputDirectory, file.Name));
        }
    }

    // event handler for Job State
    static void StateChanged(object sender, JobStateChangedEventArgs e)
    {
        Console.WriteLine("Job state changed event:");
        Console.WriteLine(" Previous state: " + e.PreviousState);
        Console.WriteLine(" Current state: " + e.CurrentState);
        switch (e.CurrentState)
        {
            case JobState.Finished:
                Console.WriteLine();
                Console.WriteLine("Job finished.");
                break;
            case JobState.Canceling:
            case JobState.Queued:
            case JobState.Scheduled:
            case JobState.Processing:
                Console.WriteLine("Please wait...\n");
                break;
            case JobState.Canceled:
                Console.WriteLine("Job is canceled.\n");
                break;
            case JobState.Error:
                Console.WriteLine("Job failed.\n");
                break;
            default:
                break;
        }
    }
}
}

```

## Sample Response (JSON)

```
{  
    "version": 1,  
    "timescale": 1000,  
    "offset": 0,  
    "framerate": 29.97,  
    "width": 1440,  
    "height": 1080,  
    "fragments": [  
        {  
            "start": 0,  
            "duration": 1067,  
            "interval": 1067,  
            "events": [  
                [  
                    {  
                        "isAdultContent": false,  
                        "adultConfidence": 0.05699,  
                        "index": 0,  
                        "timestamp": 0  
                    }  
                ]  
            ]  
        },  
        {  
            "start": 7474,  
            "duration": 1067,  
            "interval": 1067,  
            "events": [  
                [  
                    {  
                        "isAdultContent": true,  
                        "adultConfidence": 0.51886,  
                        "index": 224,  
                        "timestamp": 7474  
                    }  
                ]  
            ]  
        }  
    ]  
}
```

# Frequently Asked Questions (FAQs)

4/19/2017 • 2 min to read • [Edit Online](#)

## **What does my Content Moderator subscription include?**

Your Content Moderator subscription includes access to the review tool and the APIs. You can decide whether you want to use one or the other, or both, depending on your business needs.

## **When do I use review tool vs. the API, or both?**

**The API:** If you have an existing implementation for reviewing and/or taking action on flagged content, just use the APIs to scan your content and process the results at your end.

**The Review tool:** If you are looking to moderate your content with human review teams, while complying with your escalation workflows and content policies, the review tool is a great new product to try out. We will be constantly adding support for new content types and adding more extensibility options.

**Both:** You either have an existing API subscription that you would like to use with the review tool, or you could be trying out the review tool with one content type (images) while using your own systems for another content type (text). In any or all combinations of these scenarios, it's completely fine to use both the review tool and the API.

## **What are the limits/restrictions of the content that can be moderated by using the API?**

When using the API, images need to have a minimum of 128 pixels and a maximum file size of 4MB. Text can be at most 1024 characters long. There is no limit on the video other than those imposed by the Azure Media Service if any.

## **What happens if the content passed to the text API or the image API exceeds the size limits?**

The text API will return an error code that informs that the text is longer than permitted. The image API will also return an error code that informs that the image does not meet the size requirements.

## **Do you keep the images, text, or videos that are submitted for moderation?**

Data privacy is our top priority. Your content is your own and may not be retained by Microsoft without your consent. Microsoft uses industry-leading security measures to help protect your content.

## **Can I use Content Moderator to screen for illegal child exploitation images?**

No. However, qualified organizations can use the [PhotoDNA Cloud Service](#) to screen for this type of content.

## **What type of content (text, images, videos) can be reviewed within the review tool?**

As of now, images from automated image moderation results can be reviewed and collaborated upon within the review tool.

## **Up to how many review teams can a user join? Can the user switch between teams?**

A user can join one team at a time.

## **What kind of team member roles are supported by the review tool? How are they different?**

The Studio currently allows assigning Administrator and Reviewer roles. The Administrators can invite other users and have access to the configuration settings while reviewers can only review moderation results and tag or untag content.

## **What is a tag? Can we add our custom tags?**

A tag is a one or two-letter short code that denotes a moderation flag, such as 'a' for adult, 'r' for racy and so on. Administrators can define new tags for their business as needed.

## **How many team members can I have in my review team?**

You can have a maximum of five team members, including the administrator in a team.

# Custom Speech Service

4/12/2017 • 2 min to read • [Edit Online](#)

Welcome to Microsoft's Custom Speech Service. Custom Speech Service is a cloud-based service that provides users with the ability to customize speech models for Speech-to-Text transcription. To use the Custom Speech Service refer to the [Custom Speech Service Portal](#)

## What is the Custom Speech Service

The Custom Speech Service enables you to create customized language models and acoustic models tailored to your application and your users. By uploading your specific speech and/or text data to Custom Speech Service, you can create custom models that can be used in conjunction with Microsoft's existing state-of-the-art speech models.

For example, if you're adding voice interaction to a mobile phone, tablet or PC app, you can create a custom language model that can be combined with Microsoft's acoustic model to create a speech-to-text endpoint designed especially for your app. If your application is designed for use in a particular environment or by a particular user population, you can also create and deploy a custom acoustic model with the Custom Speech Service.

## Why use the Custom Speech Service

While the Microsoft Speech-To-Text engine is world-class, it is targeted toward the scenarios described above. However, if you expect voice queries to your application to contain particular vocabulary items, such as product names or jargon that rarely occur in typical speech, it is likely that you can obtain improved performance by customizing the language model.

For example, if you were building an app to search MSDN by voice, it's likely that terms like "object-oriented" or "namespace" or "dot net" will appear more frequently than in typical voice applications. Customizing the language model will enable the system to learn this.

## How does it work

Speech recognition systems are composed of several components that work together. Two of the most important components are the acoustic model and the language model.

The acoustic model is a classifier that labels short fragments of audio into one of a number of phonemes, or sound units, in a given language. For example, the word "speech" is comprised of four phonemes "s p iy ch". These classifications are made on the order of 100 times per second.

The language model is a probability distribution over sequences of words. The language model helps the system decide among sequences of words that sound similar, based on the likelihood of the word sequences themselves. For example, "recognize speech" and "wreck a nice beach" sound alike but the first hypothesis is far more likely to occur, and therefore will be assigned a higher score by the language model.

For more details about how to use the Custom Speech Service, refer to the [Custom Speech Service Portal](#).

- [Get Started](#)
- [FAQ](#)
- [Glossary](#)

# Get Started with Custom Speech Service

4/12/2017 • 3 min to read • [Edit Online](#)

Explore the main features of the Custom Speech Service and learn how to build, deploy and use acoustic and language models for your application needs. More extensive documentation and step-by-step instructions can be found after you sign up on the Custom Speech Services portal.

## Table of Contents

- [Prerequisites](#)
- [Step 1: Create an acoustic model](#)
- [Step 2: Create a language Model](#)
- [Step 3: Create a custom speech endpoint](#)
- [Step 4: Using a custom speech endpoint](#)

## Subscription

In order to be able to upload data, to train a model or to do a deployment you need to link your Custom Speech Service activities to an Azure subscription. This can either be a free-tier or paid tier subscription. For further information please visit the [pricing page](#). To get a subscription id please go to your Azure portal and search for cognitive services and Custom Speech Service and follow the instructions there.

The subscription key is required later in this process.

### **Subscribe to Custom Speech Service and get a subscription key**

There is a nice sample that we provide to get you going which you can find [here](#)

Before playing with the above example, you must subscribe to Custom Speech Service and get a subscription key, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial. Make sure to follow best practices for keeping your API key secret and secure.

### **Get the client library and example**

You may download a client library and example via [SDK](#). The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder.

## Step 1: Creating a custom acoustic model

To customize the acoustic model to a particular domain, a collection of speech data is required. This collection consists of a set of audio files of speech data, and a text file of transcriptions of each audio file. The audio data should be representative of the scenario in which you would like to use the recognizer

For example: If you would like to better recognize speech in a noisy factory environment, the audio files should consist of people speaking in a noisy factory. If you are interested in optimizing performance for a single speaker, e.g. you would like to transcribe all of FDR's Fireside Chats, then the audio files should consist of many examples of that speaker only.

## Step 2: Creating a custom language model

The procedure for creating a custom language model is similar to creating an acoustic model except there is no audio data, only text. The text should consist of many examples of queries or utterances you expect users to say or have logged users saying (or typing) in your application.

## Step 3: Creating a custom speech-to-text endpoint

When you have created custom acoustic models and/or language models, they can be deployed in a custom speech-to-text endpoint. To create a new custom endpoint, click "Deployments" from the "CRIS" menu on the top of the page. This takes you to a table called "Deployments" of current custom endpoints. If you have not yet created any endpoints, the table will be empty. The current locale is reflected in the table title. If you would like to create a deployment for a different language, click on "Change Locale". Additional information on supported languages can be found in the section on Changing Locale.

## Step 4: Using a custom speech endpoint

Requests can be sent to a CRIS speech-to-text endpoint in a very similar manner as the default Microsoft Cognitive Services speech endpoint. Note that these endpoints are functionally identical to the default endpoints of the Speech API. Thus, the same functionality available via the client library or REST API for the Speech API is also the available for your custom endpoint.

Please note that the endpoints created via CRIS can process different numbers of concurrent requests depending on the tier the subscription is associated to. If more recognitions than that are requested, they will return the error code 429 (Too many requests). For more information please visit the [pricing information](#). In addition, there is a monthly quota of requests for the free tier. In cases you access your endpoint in the free tier above your monthly quota the service returns the error code 403 Forbidden.

The service assumes audio is transmitted in real-time. If it is sent faster, the request will be considered running until its duration in real-time has passed.

- [Overview](#)
- [FAQ](#)
- [Glossary](#)

# Custom Speech Service Frequently Asked Questions

4/12/2017 • 6 min to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, try asking the Custom Speech Service community on [StackOverflow](#) and [UserVoice](#)

## General

**Question:** How will I know when the processing of my data set or model is complete?

**Answer:** Currently, the status of the model or data set in the table is the only want to know. When the processing is complete, the status will be "Ready". We are working on improved methods for communication processing status, such as email notification.

**Question:** Can I create more than one model at a time?

**Answer:** There is no limit to how many models are in your collection but only one can be created at a time on each page. For example, you cannot start a language model creation process if there is currently a language model in the process stage. You can, however, have an acoustic model and a langauge model processing at the same time.

**Question:** What does a status of Exception mean?

**Answer:** The Exception status indicates that something has gone wrong in the processing. If you are unable to figure out what went wrong, please contact us and we can investigate.

**Question:** I realized I made a mistake. How do I cancel my data import or model creation that's in progress?

**Answer:** Currently you cannot roll back a acoustic or language adaptation process. Imported data can be deleted after the import has been completed

**Question:** What is the difference between the Search & Dictation Models and the Conversational Models?

**Answer:** There are two Base Acoustic & Language Models to choose from in the Custom Speech Service. search queries, or dictation. The Microsoft Conversational AM is appropriate for recognizing speech spoken in a conversational style. This type of speech is typically directed at another person, such as in call centers or meetings.

**Question:** Can I update my existing model (model stacking)?

**Answer:** We do not offer the ability to update an existing model with new data. If you have a new data set and you want to customize an existing model you must re-adapt it with the new data and the old data set that you used. The old and new data sets must be combined in a single .zip (if it is acoustic data) or a .txt file if it is language data. Once adaptation is done the new updated model needs to be de-deployed to obtain a new endpoint

**Question:** What if I need higher concurrency than what either Tier1 or Tier 2 offer?

**Answer:** Tier1 offers up to 4 concurrent requests and Tier2 up to 12. Please contact us if you require higher than that.

**Question:** Can I download my model and run it locally?

**Answer:** We do not enable models to be downloaded and executed locally.

**Question:** Are my requests logged?

**Answer:** Requests are typically logged in Azure in secure storage. If you have privacy concerns that prohibit you from using the Custom Speech Service please contact us and we can discuss logging/tracing alternatives.

## Importing Data

**Question:** What is the limit on the size of the data set? Why?

**Answer:** The current limit for a data set is 2 GB, due to the restriction on the size of a file for HTTP upload.

**Question:** Can I zip my text files in order to upload a larger text file?

**Answer:** No, currently only uncompressed text files are allowed.

**Question:** The data report says there were failed utterances. Is this a problem?

**Answer:** If only a few utterances failed to be imported successfully, this is not a problem. If the vast majority of the utterances in an acoustic or language data set (e.g. >95%) are successfully imported, the data set can be usable. However, it is recommended that you try to understand why the utterances failed and fix the problems. Most common problems, such as formatting errors, are easy to fix.

## Creating AM

**Question:** How much acoustic data do I need?

**Answer:** We recommend starting with 30 minutes to one hour of acoustic data

**Question:** What sort of data should I collect?

**Answer:** You should collect data that's as close to the application scenario and use case as possible. This means the data collection should match the target application and users in terms of device or devices, environments, and types of speakers. In general, you should collect data from as broad a range of speakers as possible.

**Question:** How should I collect it?

**Answer:** You can create a standalone data collection application or use some off the shelf audio recording software. You can also create a version of your application that logs the audio data and uses that.

**Question:** Do I need to transcribe it myself?

**Answer:** The data must be transcribed. You can transcribe it yourself or use a professional transcription service. Some of these use professional transcribers and others use crowdsourcing.

**Question:** How long does it take to create a custom acoustic model?

**Answer:** The processing time for creating a custom acoustic model is about the same as the length of the acoustic data set. So, a customized acoustic model created from a five hour data set will take about five hours to process.

## Offline Testing

**Question:** Can I perform offline testing of my custom acoustic model using a custom language model?

**Answer:** Yes, just select the custom language model in the drop down when you set up the offline test

**Question:** Can I perform offline testing of my custom language model using a custom acoustic model model?

**Answer:** Yes, just select the custom acoustic model in the drop-down menu when you set up the offline test.

**Question:** What is Word Error Rate and how is it computed?

**Answer:** Word Error Rate is the evaluation metric for speech recognition. It is counted as the total number of errors, which includes insertions, deletions, and substitutions, divided by the total number of words in the reference transcription.

**Question:** I now know the test results of my custom model, is this a good or bad number?

**Answer:** The results show a comparison between the baseline model and the one you customized. You should aim to beat the baseline model to make the customization worthwhile

**Question:** How do I figure out the WER of the base models, so I can see if there was improvement?

**Answer:** The offline test results show accuracy of baseline accuracy of the custom model and the improvement over baseline

## Creating LM

**Question:** How much text data do I need to upload?

**Answer:** This is a difficult question to give a precise answer to, as it depends on how different the vocabulary and phrases used in your application are from the starting language models. For all new words, it is useful to provide as many examples as possible of the usage of those words. For common phrases that are used in your application, including those in the language data is also useful as it tells the system to listen for these terms as well. It is common to have at least one hundred and typically several hundred utterances in the language data set or more. Also if certain types \* of queries are expected to be more common than others, you can insert multiple copies of the common queries in the data set.

**Question:** Can I just upload a list of words?

**Answer:** Uploading a list of words will get the words into the vocabulary but not teach the system how the words are typically used. By providing full or partial utterances (sentences or phrases of things users are likely to say) the language model can learn the new words and how they are used. The custom language model is good not just for getting new words in the system but also for adjusting the likelihood of known words for your application. Providing full utterances helps the system learn this.

- 
- [Overview](#)
  - [Started](#)
  - [Glossary](#)

# Glossary

4/12/2017 • 1 min to read • [Edit Online](#)

## A

### **Acoustic Model**

The acoustic model is a classifier that labels short fragments of audio into one of a number of phonemes, or sound units, in a given language. For example, the word "speech" is comprised of four phonemes "s p iy ch". These classifications are made on the order of 100 times per second

## B

## C

### **Conversational Model**

A model appropriate for recognizing speech spoken in a conversational style. The Microsoft Conversational AM is adapted for speech typically directed at another person.

## D

### **Deployment**

The process through which the adapted custom model becomes a service and exposes a URI

## E

## F

## G

## H

## I

### **Inverse text normalization**

The process of converting "raw" unformatted text back to formatted text, i.e. with capitalization and punctuation, is called inverse text normalization (ITN).

## J

## K

## L

### **Language Model**

The language model is a probability distribution over sequences of words. The language model helps the system decide among sequences of words that sound similar, based on the likelihood of the word sequences themselves

M

N

### **Normalization**

Normalization (Text) : Transformation of resulting text (i.e. transcription) into a standard, unambiguous form readable by the system.

O

P

Q

R

S

### **Search and Dictate Model**

An acoustic model appropriate for processing commands. The Microsoft Search and Dictation AM is appropriate for speech directed at an application or device, such as such as commands

### **Subscription key**

Subscription key is a string that you need to specify as a query string parameter in order to invoke any Custom Speech Service model. A subscription key is obtained from [Azure Portal](#) and once obtained it can be found in "My Subscriptions" in the Custom Speech Service portal.

T

### **Transcription**

Transcription: The piece of text that results from the process of a piece of audio .wav file

U

V

W

X

Y

Z

- [Overview](#)
- [Get Started](#)
- [FAQ](#)

# Emotion API

4/12/2017 • 2 min to read • [Edit Online](#)

Welcome to the Microsoft Emotion API, which allows you to build more personalized apps with Microsoft's cutting edge cloud-based emotion recognition algorithm.

## Emotion Recognition

The Emotion API beta takes an image as an input, and returns the confidence across a set of emotions for each face in the image, as well as bounding box for the face, from the Face API. The emotions detected are happiness, sadness, surprise, anger, fear, contempt, disgust or neutral. These emotions are communicated cross-culturally and universally via the same basic facial expressions, where are identified by Emotion API.

### Interpreting Results:

In interpreting results from the Emotion API, the emotion detected should be interpreted as the emotion with the highest score, as scores are normalized to sum to one. Users may choose to set a higher confidence threshold within their application, depending on their needs.

For more details about emotion detection, please refer to the API Reference:

- Basic: If a user has already called the Face API, they can submit the face rectangle as an input and use the basic tier. [API Reference](#)
- Standard: If a user does not submit a face rectangle, they should use standard mode. [API Reference](#)

## Emotion in Video

The Emotion API for Video takes a video as an input, and returns the confidence across a set of emotions for the group of faces in the image over a period of time. The emotions detected are happiness, sadness, surprise, anger, fear, contempt, disgust or neutral. These emotions are communicated cross-culturally and universally via the same basic facial expressions, where are identified by Emotion API.

### Interpreting Results:

Emotion API for Video provides two types of aggregate results for the emotions of faces in a frame. The API first calculates emotion scores for each face in a video, smoothing the results over time for higher accuracy. It returns two types of aggregates: *windowMeanScores* gives a mean score for all of the faces detected in a frame for each emotion. The emotion detected should be interpreted as the emotion with the highest score, as scores are normalized to sum to one. Users may choose to set a higher confidence threshold within their application, depending on their needs. *windowFaceDistribution* gives the distribution of faces with each emotion as the dominant emotion for that face. Dominant emotions for each face have been determined based on the emotion with the highest score for that face.

Because emotions are smoothed over time, if you ever build a visualization to overlay your results on top of the original video, subtract 250 milliseconds from the provided timestamps.

For more detail on how to parse the format of Emotion for Video API results, you may also want to view For more details about emotion detection in video, please refer to the [API Reference](#).

# How to Call Emotion API for Video

4/12/2017 • 4 min to read • [Edit Online](#)

This guide demonstrates how to call Emotion API for Video. The samples are written in C# using the Emotion API for Video client library.

In order to use the Emotion API for Video, you will need a video that includes people, preferably video where the people are facing the camera.

Every call to the Emotion API for Video requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header. To pass the subscription key through a query string, refer to the request URL below for the Emotion API for Video as an example:

```
https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognizeInVideo&subscription-key=<Your subscription key>
```

As an alternative, the subscription key can also be specified in the HTTP request header:

```
ocp-apim-subscription-key: <Your subscription key>
```

When using a client library, the subscription key is passed in through the constructor of the `VideoServiceClient` class. For example:

```
var emotionServiceClient = new emotionServiceClient("Your subscription key");
```

To obtain a subscription key, see [Subscriptions](#).

The most basic way to perform any of the Emotion API for Video calls is by uploading a video directly. This is done by sending a "POST" request with application/octet-stream content type together with the data read from a video file. The maximum size of the video is 100MB.

Using the client library, stabilization by means of uploading is done by passing in a stream object. See the example below:

```
Operation videoOperation;
using (var fs = new FileStream(@"C:\Videos\Sample.mp4", FileMode.Open))
{
    videoOperation = await videoServiceClient.CreateOperationAsync(fs, OperationType.recognizeInVideo);
}
```

Please note that the `CreateOperationAsync` method of `VideoServiceClient` is `async`. The calling method should be marked as `async` as well in order to use the `await` clause. If the video is already on the web and has a public URL, Emotion API for Video can be accessed by providing the URL. In this example, the request body will be a JSON string which contains the URL.

Using the client library, stabilization by means of an URL can easily be executed using another overload of the `CreateOperationAsync` method.

```
var videoUrl = "http://www.example.com/sample.mp4";
Operation videoOperation = await videoServiceClient.CreateOperationAsync(videoUrl, OperationType.
recognizeInVideo);
```

This upload method will be the same for all the Emotion API for Video calls.

Once you have uploaded a video, the next operation you will want to perform is to check its status. Because video files are usually larger and more diverse than other files, users can expect a long processing time at this step. The time depends on the size and the length of the file.

Using the client library, you can retrieve the operation status and result using the GetOperationResultAsync method.

```
var operationResult = await videoServiceClient.GetOperationResultAsync(videoOperation);
```

Typically, the client side should periodically retrieve the operation status until the status is shown as "Succeeded" or "Failed".

```
OperationResult operationResult;
while (true)
{
    operationResult = await videoServiceClient.GetOperationResultAsync(videoOperation);
    if (operationResult.Status == OperationStatus.Succeeded || operationResult.Status ==
OperationStatus.Failed)
    {
        break;
    }

    Task.Delay(30000).Wait();
}
```

When the status of VideoOperationResult is shown as "Succeeded" the result can be retrieved by casting the VideoOperationResult to a VideoOperationInfoResult and accessing the ProcessingResult field.

```
var emotionRecognitionJsonString =
((VideoOperationInfoResult<VideoAggregateRecognitionResult>)operationResult).ProcessingResult;
```

The `emotionRecognitionJsonString` contains the metadata from the faces within the given file in JSON format.

As explained in Step 2, the JSON output is available in the ProcessingResult field of OperationResult, when its status is shown as "Succeeded".

The face detection and tracking JSON includes the following attributes:

ATTRIBUTE	DESCRIPTION
Version	Refers to the version of the Emotion API for Video JSON.
Timescale	"Ticks" per second of the video.
Offset	The time offset for timestamps. In version 1.0 of Emotion API for Videos, this will always be 0. In future supported scenarios, this value may change.
Framerate	Frames per second of the video.

ATTRIBUTE	DESCRIPTION
Fragments	The metadata is cut up into different smaller pieces called fragments. Each fragment contains a start, duration, interval number, and event(s).
Start	The start time of the first event, in ticks.
Duration	The length of the fragment, in ticks.
Interval	The length of each event within the fragment, in ticks.
Events	An array of events. The outer array represents one interval of time. The inner array consists of 0 or more events that happened at that point in time.
windowFaceDistribution	Percent faces to have a particular emotion during the event.
windowMeanScores	Mean scores for each emotion of the faces in the image.

The reason for formatting the JSON this way is to set up the APIs for future scenarios, where it will be important to retrieve metadata quickly and manage a large stream of results. This formatting is using both the techniques of fragmentation (allowing you to break up the metadata in time-based portions, where you can download only what you need), and segmentation (allowing you to break up the events if they get too large). Some simple calculations can help you transform the data. For example, if an event started at 6300 (ticks), with a timescale of 2997 (ticks/sec) and a framerate of 29.97 (frames/sec), then:

- Start/Timescale = 2.1 seconds
- Seconds x (Framerate/Timescale) = 63 frames

Below is a simple example of extracting the JSON into a per frame format for face detection and tracking:

```
var emotionRecognitionTrackingResultJsonString = operationResult.ProcessingResult;
var emotionRecognitionTracking = JsonConvert.DeserializeObject<EmotionRecognitionResult>
(emotionRecognitionTrackingResultJsonString, settings);
```

Because emotions are smoothed over time, if you ever build a visualization to overlay your results on top the original video, subtract 250 milliseconds from the provided timestamps.

In this guide you have learned about the functionalities of the Emotion API for Video: how you can upload a video, check its status, retrieve emotion recognition metadata.

For more information about API details, see the API reference guide "[Emotion API for Video Reference](#)".

# How to Analyze Videos in Real-time

4/12/2017 • 7 min 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 uses 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();

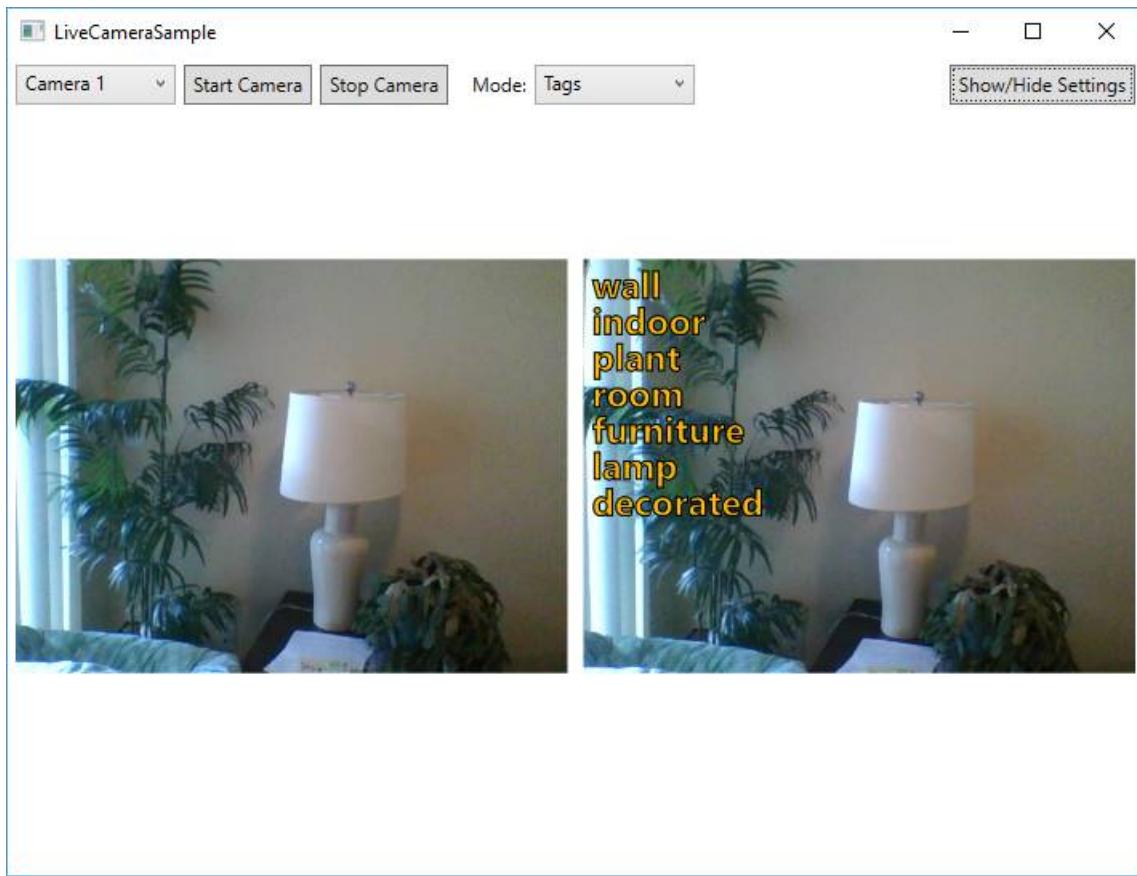
            // Wait for keypress to stop
            Console.WriteLine("Press any key to stop...");
            Console.ReadKey();

            // 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 [microsoft.com/cognitive](https://microsoft.com/cognitive). 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.

To report abuse of the Microsoft Cognitive Services to Microsoft, please visit the [Microsoft Cognitive Services website](#), and use the "Report Abuse" link at the bottom of the page to contact Microsoft. For more information about Microsoft privacy policies please see their privacy statement here: <https://go.microsoft.com/fwlink/?LinkId=521839>.

## 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](#).

# Emotion API cURL Quick Start

4/12/2017 • 1 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the [Emotion API Recognize method](#) with cURL to recognize the emotions expressed by one or more people in an image.

## Prerequisite

- Get your free Subscription Key [here](#)

## Recognize Emotions cURL Example Request

```
@ECHO OFF

curl -v -X POST "https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"
```

## Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- **faceRectangle** - Rectangle location of face in the image.
- **scores** - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

# Emotion API C# Quick Start

4/12/2017 • 1 min to read • [Edit Online](#)

This article provides information and a code sample to help you quickly get started using the [Emotion API Recognize method](#) with C# to recognize the emotions expressed by one or more people in an image.

## Prerequisites

- Get the Microsoft Cognitive Emotion API Windows SDK [here](#)
- Get your free Subscription Key [here](#)

## Emotion Recognition C# Example Request

```

using System;
using System.IO;
using System.Net.Http.Headers;
using System.Net.Http;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter the path to a JPEG image file:");
            string imagePath = Console.ReadLine();

            MakeRequest(imagePath);

            Console.WriteLine("\n\n\nWait for the result below, then hit ENTER to exit...\n\n\n");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void MakeRequest(string imagePath)
        {
            var client = new HttpClient();

            // Request headers
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "40d7576afa7e4f82b38315516d55cb5e");

            string uri = "https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize?";
            HttpResponseMessage response;
            string responseContent;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);

            using (var 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");
                response = await client.PostAsync(uri, content);
                responseContent = response.Content.ReadAsStringAsync().Result;
            }

            //A peak at the JSON response.
            Console.WriteLine(responseContent);
        }
    }
}

```

## Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- **faceRectangle** - Rectangle location of face in the image.
- **scores** - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

# Emotion API Java for Android Quick Start

4/12/2017 • 1 min to read • [Edit Online](#)

This article provides information and a code sample to help you quickly get started with the [Emotion Recognize method](#) in the Emotion API Android client library. The sample demonstrates how you can use Java to recognize the emotions expressed by people.

## Prerequisites

- Get the Emotion API Java for Android SDK [here](#)
- Get your free subscription key [here](#)

## Recognize Emotions Java for Android Example Request

```

// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
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.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            URIBuilder uriBuilder = new
URIBuilder("https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize");

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

            // Request headers. Replace the example key below with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. Replace the example URL below with the URL of the image you want to analyze.
            StringEntity reqEntity = new StringEntity("{ \"url\": \"http://example.com/images/test.jpg\" }");
            request.setEntity(reqEntity);

            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                System.out.println(EntityUtils.toString(entity));
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

## Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- **faceRectangle** - Rectangle location of face in the image.
- **scores** - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

# Emotion API JavaScript Quick Start

4/12/2017 • 1 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the [Emotion API Recognize method](#) with JavaScript to recognize the emotions expressed by one or more people in an image.

## Prerequisite

- Get your free Subscription Key [here](#)

## Recognize Emotions JavaScript Example Request

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

<script type="text/javascript">
$(function() {
    var params = {
        // Request parameters
    };

    $.ajax({
        url: "https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize?" + $.param(params),
        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","{subscription key}");
        },
        type: "POST",
        // Request body
        data: "{body}",
    })
    .done(function(data) {
        alert("success");
    })
    .fail(function() {
        alert("error");
    });
});
</script>
</body>
</html>
```

## Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- **faceRectangle** - Rectangle location of face in the image.
- **scores** - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

# Emotion API PHP Quick Start

4/12/2017 • 1 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using PHP and the [Emotion API Recognize method](#) to recognize the emotions expressed by one or more people in an image.

## Prerequisite

- Get your free Subscription Key [here](#)

## Recognize Emotions PHP Example Request

```
<?php
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)
require_once 'HTTP/Request2.php';

$request = new Http_Request2('https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize');
$url = $request->getUrl();

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

$parameters = array(
    // Request parameters
);

$url->setQueryVariables($parameters);

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

// Request body
$request->setBody("{body}");

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>
```

## Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- faceRectangle - Rectangle location of face in the image.
- scores - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

# Emotion API Python Quick Start

4/12/2017 • 1 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the [Emotion API Recognize method](#) with Python to recognize the emotions expressed by one or more people in an image.

## Prerequisite

- Get your free Subscription Key [here](#)

## Recognize Emotions Python Example Request

```

##### Python 2.7 #####
import httplib, urllib, base64

headers = {
    # Request headers. Replace the placeholder key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.urlencode({
})

# Replace the example URL below with the URL of the image you want to analyze.
body = "{ 'url': 'http://example.com/picture.jpg' }"

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/emotion/v1.0/recognize?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64, sys

headers = {
    # Request headers. Replace the placeholder key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

params = urllib.parse.urlencode({
})

# Replace the example URL below with the URL of the image you want to analyze.
body = "{ 'url': 'http://example.com/picture.jpg' }"

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/emotion/v1.0/recognize?%s" % params, body, headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print(e.args)
#####

```

## Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- **faceRectangle** - Rectangle location of face in the image.
- **scores** - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

title: Emotion API Ruby quick start | Microsoft Docs description: Get information and code samples to help you quickly get started using the Emotion API with Ruby in Cognitive Services. services: cognitive-services author: v-royhar manager: yutkuo

ms.service: cognitive-services ms.technology: emotion ms.topic: article ms.date:01/30/2017

ms.author: anroth

## Emotion API Ruby Quick Start

This article provides information and code samples to help you quickly get started using the [Emotion API Recognize method](#) with Ruby to recognize the emotions expressed by one or more people in an image.

### Prerequisite

- Get your free Subscription Key [here](#)

### Recognize Emotions Ruby Example Request

```
require 'net/http'

uri = URI('https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize')
uri.query = URI.encode_www_form({})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# Request headers
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Request body
request.body = "{body}"

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

puts response.body
```

### Recognize Emotions Sample Response

A successful call returns an array of face entries and their associated emotion scores, ranked by face rectangle size in descending order. An empty response indicates that no faces were detected. An emotion entry contains the following fields:

- faceRectangle - Rectangle location of face in the image.
- scores - Emotion scores for each face in the image.

```
application/json
[
  {
    "faceRectangle": {
      "left": 68,
      "top": 97,
      "width": 64,
      "height": 97
    },
    "scores": {
      "anger": 0.00300731952,
      "contempt": 5.14648448E-08,
      "disgust": 9.180124E-06,
      "fear": 0.0001912825,
      "happiness": 0.9875571,
      "neutral": 0.0009861537,
      "sadness": 1.889955E-05,
      "surprise": 0.008229999
    }
  }
]
```

# Emotion API in C# Tutorial

4/12/2017 • 5 min to read • [Edit Online](#)

Explore a basic Windows application that uses Emotion API to recognize the emotions expressed by the 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 Emotion API and WPF (Windows Presentation Foundation), a part of .NET Framework.

## Platform requirements

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

## Subscribe to Emotion API and get a subscription key

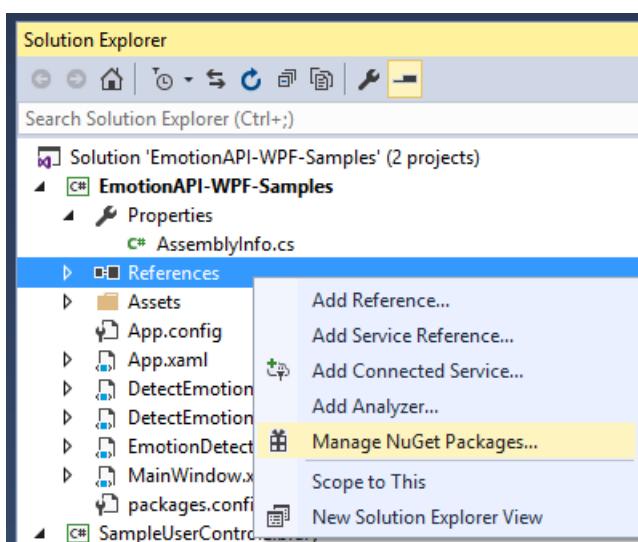
Before creating the example, you must subscribe to Emotion API which is part of the Microsoft Cognitive Services (previously Project Oxford). See [Subscriptions](#). Both the primary and secondary key can be used in this tutorial. Make sure to follow best practices for keeping your API key secret and secure.

## Get the client library and example

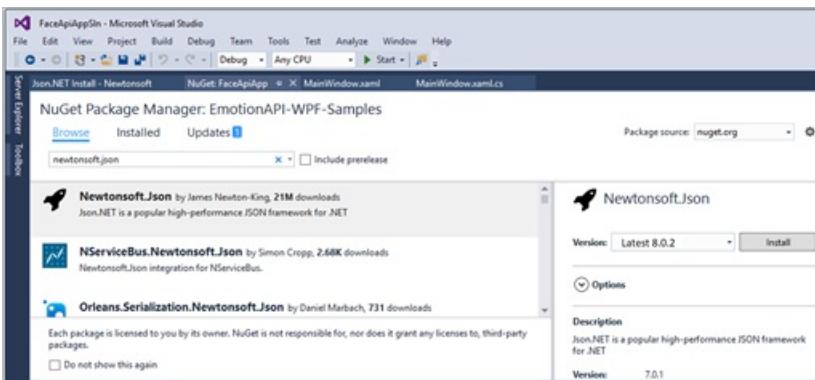
You may download the Emotion API client library via [SDK](#). The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder.

1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Browse to the folder where you saved the downloaded Emotion API files. Click on **Emotion**, then **Windows**, and then the **Sample-WPF** folder.
3. Double-click to open the Visual Studio 2015 Solution (.sln) file named **EmotionAPI-WPF-Samples.sln**. This will open the solution in Visual Studio.

4. In **Solution Explorer**, right click **References** and select **Manage NuGet Packages**.



2. The **NuGet Package Manager** window opens. First select **Browse** in the upper left corner, then in the search box type "Newtonsoft.Json", select the **Newtonsoft.Json** package and click **Install**.



3. 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 Emotion API window with the **text box** reading "**Paste your subscription key here to start**". Paste your subscription key into the text box as shown in below screenshot. 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 either of the two scenarios, "**Detect emotion using a stream**" or "**Detect emotion using a URL**", then follow the instructions on the screen. Microsoft receives the images you upload and may use them to improve Emotion API and related services. By submitting an image, you confirm that you have followed our [Developer Code of Conduct](#).
4. There are example images to be used with this example application. You can find these images on [the Face API Github repo](#) under the **Data** folder. Please note the use of these images is licensed under Fair Use agreement meaning they are OK to use for testing this example, but not for republishing.

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

This example app makes use of the Emotion API Client Library, a thin C# client wrapper for the Microsoft Emotion 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 **Emotion, 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 **EmotionAPI-WPF\_Samples**, expand **DetectEmotionUsingStreamPage.xaml** to locate **DetectEmotionUsingStreamPage.xaml.cs**, which is used for browsing to a locally stored file, or expand **DetectEmotionUsingURLPage.xaml** to find **DetectEmotionUsingURLPage.xaml.cs**, which is used when uploading an image URL. Double-click the .xaml.cs files to have them open in new windows in Visual Studio.

Reviewing how the Emotion Client Library gets used in our example app, let's look at two code snippets from **DetectEmotionUsingStreamPage.xaml.cs** and **DetectEmotionUsingURLPage.xaml.cs**. Each 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 Emotion API is able to work with either an image URL or binary image data (in form of an octet stream) as

input. The two options are reviewed below. In both cases, you first find a using directive, which lets you use the Emotion Client Library.

```
// -----
// KEY SAMPLE CODE STARTS HERE
// Use the following namespace for EmotionServiceClient
// -----
using Microsoft.ProjectOxford.Emotion;
using Microsoft.ProjectOxford.Emotion.Contract;
// -----
// KEY SAMPLE CODE ENDS HERE
// -----
```

#### DetectEmotionUsingURLPage.xaml.cs

This code snippet shows how to use the Client Library to submit your subscription key and a photo URL to the Emotion API service.

```
// -----
// KEY SAMPLE CODE STARTS HERE
// -----
window.Log("EmotionServiceClient is created");

//
// Create Project Oxford Emotion API Service client
//
EmotionServiceClient emotionServiceClient = new EmotionServiceClient(subscriptionKey);

window.Log("Calling EmotionServiceClient.RecognizeAsync()...");
try
{
    //
    // Detect the emotions in the URL
    //
    Emotion[] emotionResult = await emotionServiceClient.RecognizeAsync(url);
    return emotionResult;
}
catch (Exception exception)
{
    window.Log("Detection failed. Please make sure that you have the right subscription key and
proper URL to detect.");
    window.Log(exception.ToString());
    return null;
}
// -----
// KEY SAMPLE CODE ENDS HERE
// -----
```

#### DetectEmotionUsingStreamPage.xaml.cs

Shown below is how to submit your subscription key and a locally stored image to the Emotion API.

```
// -----
// KEY SAMPLE CODE STARTS HERE
// -----



//
// Create Project Oxford Emotion API Service client
//
EmotionServiceClient emotionServiceClient = new EmotionServiceClient(subscriptionKey);

window.Log("Calling EmotionServiceClient.RecognizeAsync()...");

try
{
    Emotion[] emotionResult;
    using (Stream imageFileStream = File.OpenRead(imageFilePath))
    {
        //
        // Detect the emotions in the URL
        //
        emotionResult = await emotionServiceClient.RecognizeAsync(imageFileStream);
        return emotionResult;
    }
}
catch (Exception exception)
{
    window.Log(exception.ToString());
    return null;
}
// -----
// KEY SAMPLE CODE ENDS HERE
// -----
```

# Emotion API using Python Tutorial

4/12/2017 • 1 min to read • [Edit Online](#)

To make it easy to get started with Emotion API, the Jupyter notebook linked below shows you how to use the API in Python and how to visualize your results using some popular libraries.

[Link to notebook in GitHub](#)

## Using the Jupyter Notebook

To use the notebook interactively, you will need to clone it and run it in Jupyter. To learn how to get started with interactive Jupyter notebooks, follow the instructions at <http://jupyter.readthedocs.org/en/latest/install.html>.

To use this notebook, you will need a subscription key for the Emotion 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 shown below. Either the primary or the secondary key works.

Python Example

```
#Variables  
  
_url = 'https://westus.api.cognitive.microsoft.com/emotion/v1.0/recognize'  
_key = None #Here you have to paste your primary key  
_maxNumRetries = 10
```

# Emotion API Frequently Asked Questions

4/12/2017 • 1 min to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, try asking the Emotion API community on [StackOverflow](#) or contact Help and Support on [UserVoice](#).

**Question:** *What types of images get the best results from Emotion API?*

**Answer:** Use unobstructed, full frontal facial images for best results. Reliability decreases with partial frontal faces and Emotion API may not recognize emotions in images where the face is rotated 45 degrees or more.

**Question:** *How many emotions can Emotion API identify?*

**Answer:** Emotion API recognizes eight different universally-accepted emotions:

- Happiness
- Sadness
- Surprise
- Anger
- Fear
- Contempt
- Disgust
- Neutral

**Question:** *Is there any way to pass a live video stream to the API and get the result simultaneously?*

**Answer:** Use the image based emotion API and call it on each frame or skip frames for performance. Video Frame-by-Frame Analysis samples are available.

**Question:** *I am passing the binary image data in but it gives me: "Invalid face image.\**

**Answer:** This implies that the algorithm had an issue parsing the image.

- The supported input image formats includes JPEG, PNG, GIF(the first frame), BMP.
- Image file size should be no larger than 4MB
- The detectable face size range is 36x36 to 4096x4096 pixels. Faces out of this range will not be detected
- Some faces may not be detected due to technical challenges, e.g. very large face angles (head-pose), large occlusion. Frontal and near-frontal faces have the best results

# Entity Linking Intelligence Service API

4/12/2017 • 1 min to read • [Edit Online](#)

Welcome to the Microsoft Entity Linking Intelligence Service, a web service created to help developers with tasks relating to entity linking.

## Entity Linking

Sometimes in different contexts, a word might be used as a named entity, a verb, or other word form within a given sentence. For example, in the case where "times" is a named entity, it still may refer to two separately distinguishable entities, such as "The New York Times" or "Times Square". Given a specific paragraph within a document, the Entity Linking Intelligence Service will recognize and identify each separate entity based on its context.

The illustration below shows an example of entity linking. Specifically, when using Wikipedia, the Entity Linking Intelligence Service detects all entities mentioned within the input text and links them to relevant reference points according to the page ID.



## Get Started

To quickly go through the Entity Linking basic functionalities and subscription process, refer to our getting started tutorial. [Getting Started with Entity Linking API in C#](#)

# Get Started with Entity Linking API in C#

4/12/2017 • 3 min to read • [Edit Online](#)

Microsoft's Entity Linking is a natural language processing tool to analyze text and link named-entities to relevant entries in a knowledge base.

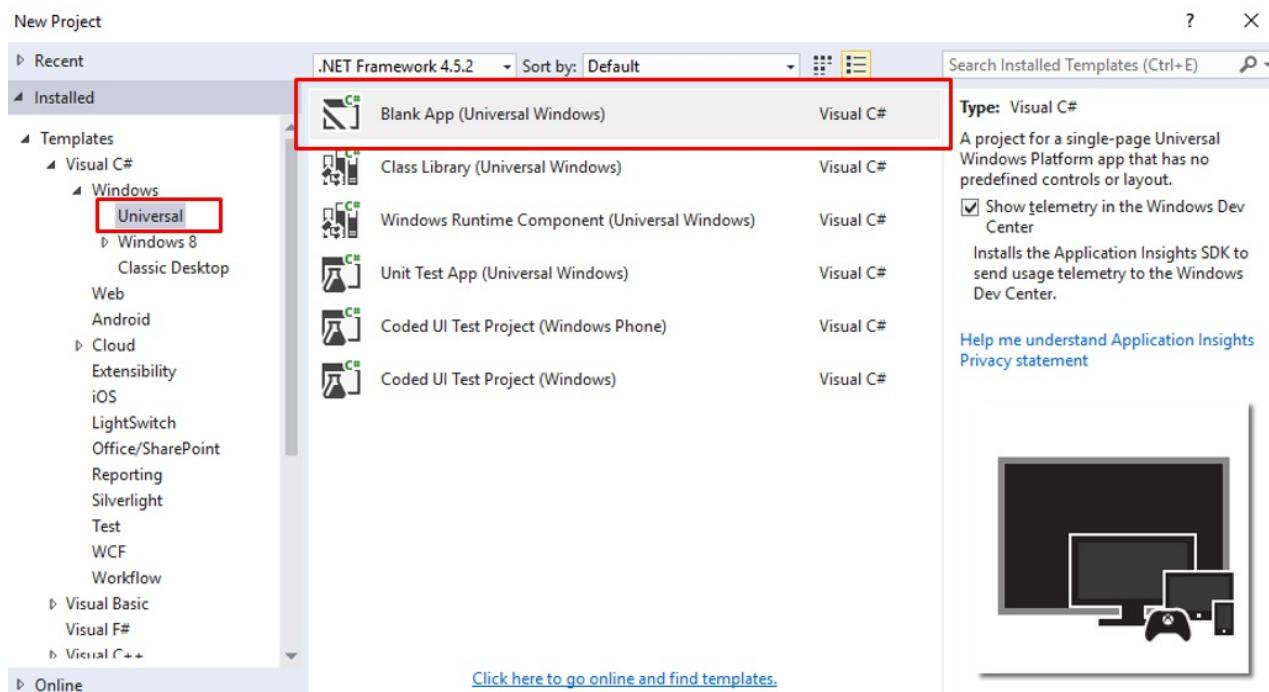
This tutorial explores entity linking by using the Entity Linking Client Library as a NuGet package.

- Visual Studio 2015
- A Microsoft Cognitive Services API Key
- Get the client library and example
- Microsoft Entity Linking NuGet Package

You may download the Entity Linking Intelligence Service API Client Library via [SDK](#). The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder.

Before using Entity Linking Intelligence Service, you must sign up for an API key. See [Subscriptions](#). Both the primary and secondary key can be used in this tutorial.

Let's start by creating a new project in Visual Studio. First, launch Visual Studio 2015 from the Start Menu. Then, create a new project by selecting **Installed → Templates → Visual C# → Windows Universal → Blank App** for your project template:



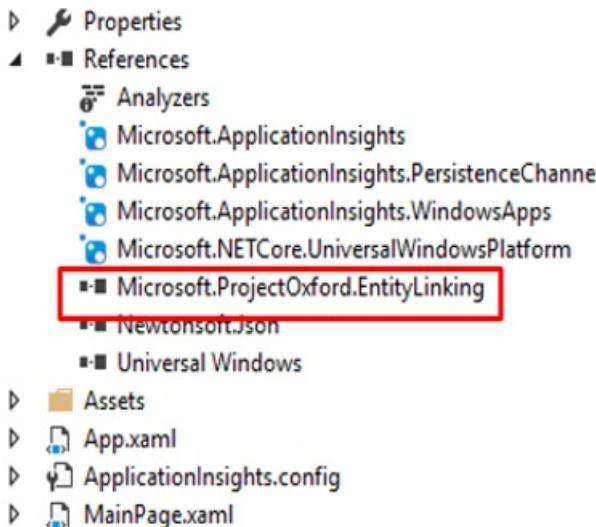
Entity Linking of Cognitive Services is released as a NuGet.org package and needs to be installed before you can use it. To add it to your project, go to the **Solution Explorer** tab, right click your project, and select **Manage Nuget Packages**.

First, in the **NuGet Package Manager** window, select NuGet.org as your **Package Source** in the upper right corner. Select **Browse** in the upper left corner and in the search box type "ProjectOxford.EntityLinking". Select the **Microsoft.ProjectOxford.EntityLinking** NuGet package and click **Install**.

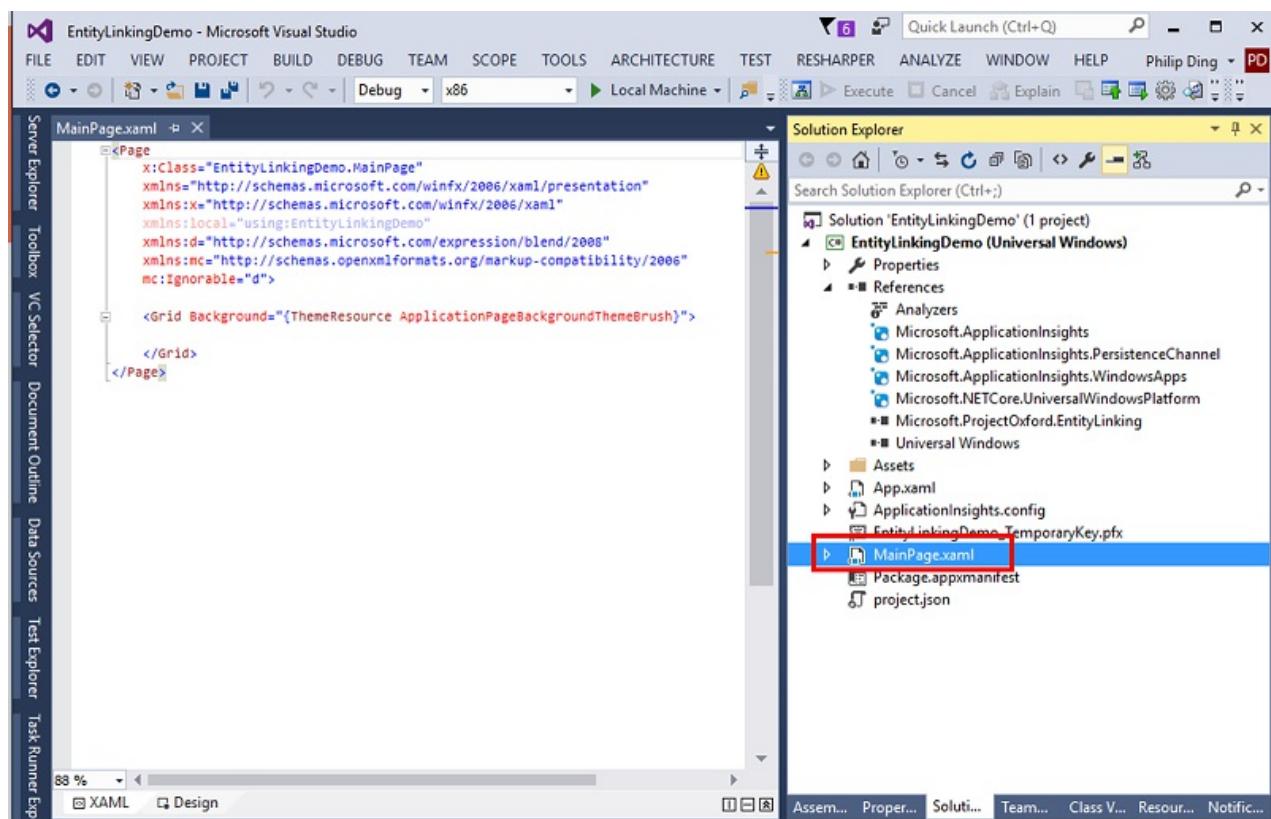
Next, search for Newtonsoft.Json and install. If you are prompted to review changes, click **OK**. If you are presented

with the EntityLinking license terms, click **I Accept**.

Entity Linking is now installed as part of your application. You can confirm this by checking that the\*\* Microsoft.ProjectOxford.EntityLinking\*\* reference is present as part of your project in Solution Explorer.



Navigate to \*\* MainPage.xaml \*\* in **Solution Explorer**, then double click the file which will open it in a new window. For convenience, you can double click on the **XAML** button in the **Designer** tab, this will hide the **Visual Designer** and reserve all of the space for the code view.



As a text service, the best way to visualize the functionality is creating an input and an output text block. To do this, add the following XAML in the **Grid**. This code adds three components, an input text box, an output text block, and a start button.

```

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="*"/>
        <RowDefinition Height="*" />
        <RowDefinition Height="50" />
    </Grid.RowDefinitions>
    <TextBox x:Name="inputBox" Grid.Row="0" TextWrapping="Wrap" Text="Enter a paragraph" Margin="10" AcceptsReturn="True" />
    <TextBlock x:Name="outputBlock" Grid.Row="1" TextWrapping="Wrap" Text="Result will be here" Margin="10" />
    <Button x:Name="button" Grid.Row="2" Content="Get Result" />
</Grid>

```

The user interface is now created. Before using the Entity Linking service, we need to add the button-Click handler. Open **MainPage.xaml** from **Solution Explorer**. Add a button\_Click handler in the end of the button.

```
<Button x:Name="button" Grid.Row="2" Content="Get Result" Click="button_Click" />
```

A button-Click handler needs to be implemented in the code. Open **MainPage.xaml.cs** from **Solution Explorer** to implement the button-Click. The EntityLinkingServiceClient is a wrapper to retrieve Entity Linking responses. The constructor argument of EntityLinkingServiceClient is the Cognitive Services subscription key. Paste in the subscription key you got in **Step 1** to call the Entity Linking service.

Below is example code, which adds the "wikipediaID" to the response by using Entity Linking Service.

```

private async void button_Click(object sender, RoutedEventArgs e)
{
    var text = this.inputBox.Text;
    var client = new EntityLinkingServiceClient("Your subscription key");
    var linkResponse = await client.LinkAsync(text);
    var result = string.Join(", ", linkResponse.Select(i => i.WikipediaID).ToList());
    this.outputBlock.Text = result;
}

```

Now you are ready to run your first natural language processing Entity Linking App. Press the **F5 key** to compile and launch the application. Paste text snippets or paragraphs into the input box. Press the "Get Result" button and observe the identified entities in the output block.

A year ago we heard that NASA and Microsoft were teaming up to build Sidekick, a project that uses HoloLens to let astronauts and scientists collaborate remotely, as well as visualize 3D schematics. Now we've finally got a closer look at Sidekick in action thanks to NASA's Jeff Norris, who discussed the project during a Vision Summit presentation. Norris, who leads mission control innovation for NASA's Jet Propulsion Laboratory, mainly focuses on the 3D visualization aspect ("Procedure Mode").

Rather than waiting for components to be assembled for testing, Sidekick lets NASA explore potential issues ahead of time. Engineers and scientists can do everything from view full-scale holograms to dive into individual components. The sample imagery in the video doesn't look too complex at the moment, but it's the sort of thing that will evolve along with Microsoft's HoloLens hardware.

NASA, Microsoft, Visualization (computer graphics), Jet Propulsion Laboratory

Get Result

In this tutorial you've learned how to create an application to leverage Entity Linking Intelligence Service Client Library with just a few lines of C# and XAML code.

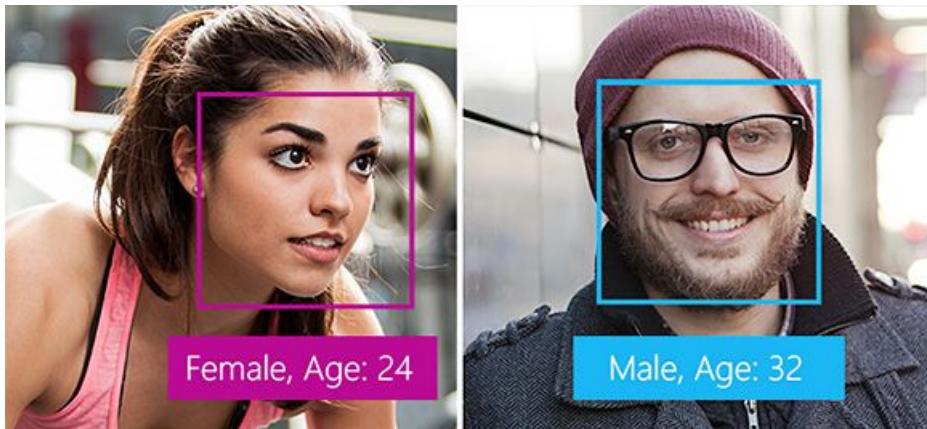
# Face API

4/12/2017 • 3 min to read • [Edit Online](#)

Welcome to the Microsoft Face API, a cloud-based service that provides the most advanced face algorithms. Face API has two main functions: face detection with attributes and face recognition.

## Face Detection

Face API detects up to 64 human faces with high precision face location in an image. And the image can be specified by file in bytes or valid URL.



Face rectangle (left, top, width and height) indicating the face location in the image is returned along with each detected face. Optionally, face detection extracts a series of face related attributes such as pose, gender, age, head pose, facial hair and glasses. Refer to [Face - Detect](#) for more details.

## Face Recognition

Face recognition is widely used in many scenarios including security, natural user interface, image content analysis and management, mobile apps, and robotics. Four face recognition functions are provided: face verification, finding similar faces, face grouping, and person identification.

### Face Verification

Face API verification performs an authentication against two detected faces or authentication from one detected face to one person object. Refer to [Face - Verify](#) for more details.

### Finding Similar Face

Given target detected face and a set of candidate faces to search with, our service finds a small set of faces that look most similar to the target face. Two working modes, `matchFace` and `matchPerson` are supported. `matchPerson` mode returns similar faces after applying a same-person threshold derived from [Face - Verify](#). `matchFace` mode ignores the same-person threshold and returns top similar candidate faces. Take the following example, candidate faces are listed.



And query face is



To find 4 similar faces, `matchPerson` mode returns (a) and (b), which belong to the same person with query face. `matchFace` mode returns (a), (b), (c) and (d), exactly 4 candidates even if low similarity. Refer to [Face - Find Similar](#) for more details.

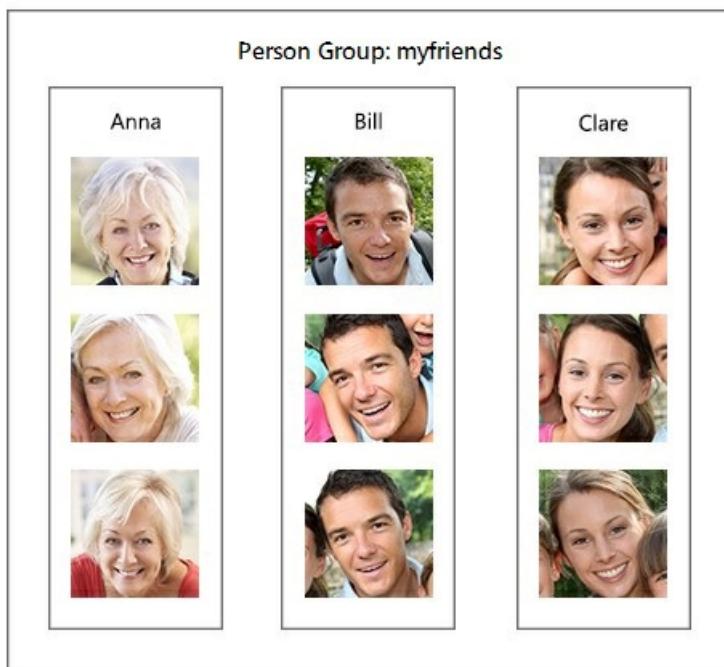
## Face Grouping

Given one set of unknown faces, face grouping API automatically divides them into several groups based on similarity. Each group is a disjointed proper subset of the original unknown face set, and contains similar faces. And all the faces in the same group can be considered to belong to the same person object. Refer to [Face - Group](#) for more details.

## Face Identification

Face API can be used to identify people based on a detected face and people database (defined as a person group) which needs to be created in advance and can be edited over time.

The following figure is an example of a person group named "myfriends". Each group may contain up to 1,000 person objects. Meanwhile, each person object can have one or more faces registered.



After a person group has been created and trained, identification can be performed against the group and a new detected face. If the face is identified as a person object in the group, the person object will be returned.

For more details about person identification, please refer to the API guides listed below:

[Face - Identify](#)

[Person Group - Create a Person Group](#)

[Person - Create a Person](#)

## Person Group - Train Person Group

### Face Storage

Face Storage allows a Standard subscription to store additional persisted faces when using Person objects ([Person - Add A Person Face](#)) or Face Lists ([Face List - Add a Face to a Face List](#)) for identification or similarity matching with the Face API. The stored images are charged at \$0.5 per 1000 faces and this rate is prorated on a daily basis. Free tier subscriptions are free, but limited to 1,000 total persons.

Pricing for Face Storage is prorated daily. For example, if your account used 10,000 persisted faces each day for the first half of the month and none the second half, you would be billed only for the 10,000 faces for the days stored. Alternatively, if each day during the month you persist 1,000 faces for a few hours and then delete them each night, you would still be billed for 1,000 persisted faces each day.

## Getting Started Tutorials

The following tutorials demonstrate the Face API basic functionalities and subscriptions processes:

- [Getting Started with Face API in CSharp Tutorial](#)
- [Getting Started with Face API in Java for Android Tutorial](#)
- [Getting Started with Face API in Python Tutorial](#)

## Sample Apps

Take a look at these sample applications which make use of Face API.

- [FamilyNotes UWP app](#)
  - Universal Windows Platform (UWP) sample app that shows usage of speech, Cortana, ink, and camera through a family note sharing scenario.
- [Video Frame Analysis Sample](#)
  - Universal Windows Platform (UWP) sample app that shows analyzing live video streams in near real-time using the Face, Computer Vision, and Emotion APIs.

## Related Topics

- [Face API Version 1.0 Release Notes](#)
- [How to Detect Faces in Image](#)
- [How to Identify Faces in Image](#)

# How to Detect Faces in Image

4/12/2017 • 5 min to read • [Edit Online](#)

This guide will demonstrate how to detect faces from an image, with face attributes like gender, age, or pose extracted. The samples are written in C# using the Face API client library.

## Concepts

If you are not familiar with any of the following concepts in this guide, please refer to the definitions in our [Glossary](#) at any time:

- Face detection
- Face landmarks
- Head pose
- Face attributes

## Preparation

In this sample, we will demonstrate the following features:

- Detecting faces from an image, and marking them using rectangular framing
- Analyzing the locations of pupils, the nose or mouth, and then marking them in the image
- Analyzing the head pose, gender and age of the face

In order to execute these features, you will need to prepare an image with at least one clear face.

## Step 1: Authorize the API call

Every call to the Face API requires a subscription key. This key needs to be either passed through a query string parameter, or specified in the request header. To pass the subscription key through query string, please refer to the request URL for the [Face - Detect](#) as an example:

```
https://westus.api.cognitive.microsoft.com/face/v1.0/detect[?returnFaceId][&returnFaceLandmarks]
[&returnFaceAttributes]
&subscription-key=<Your subscription key>
```

As an alternative, the subscription key can also be specified in the HTTP request header: **ocp-apim-subscription-key: <Your subscription key>** When using a client library, the subscription key is passed in through the constructor of the FaceServiceClient class. For example:

```
faceServiceClient = new FaceServiceClient("Your subscription key");
```

See [Subscription and key management](#).

## Step 2: Upload an image to the service and execute face detection

The most basic way to perform face detection is by uploading an image directly. This is done by sending a "POST" request with application/octet-stream content type, with the data read from a JPEG image. The maximum size of the image is 4MB.

Using the client library, face detection by means of uploading is done by passing in a Stream object. See the example below:

```
using (Stream s = File.OpenRead(@"D:\MyPictures\image1.jpg"))
{
    var faces = await faceServiceClient.DetectAsync(s, true, true);

    foreach (var face in faces)
    {
        var rect = face.FaceRectangle;
        var landmarks = face.FaceLandmarks;
    }
}
```

Please note that the DetectAsync method of FaceServiceClient is async. The calling method should be marked as async as well, in order to use the await clause. If the image is already on the web and has an URL, face detection can be executed by also providing the URL. In this example, the request body will be a JSON string which contains the URL. Using the client library, face detection by means of a URL can be executed easily using another overload of the DetectAsync method.

```
string imageUrl = "http://news.microsoft.com/ceo/assets/photos/06_web.jpg";
var faces = await faceServiceClient.DetectAsync(imageUrl, true, true);

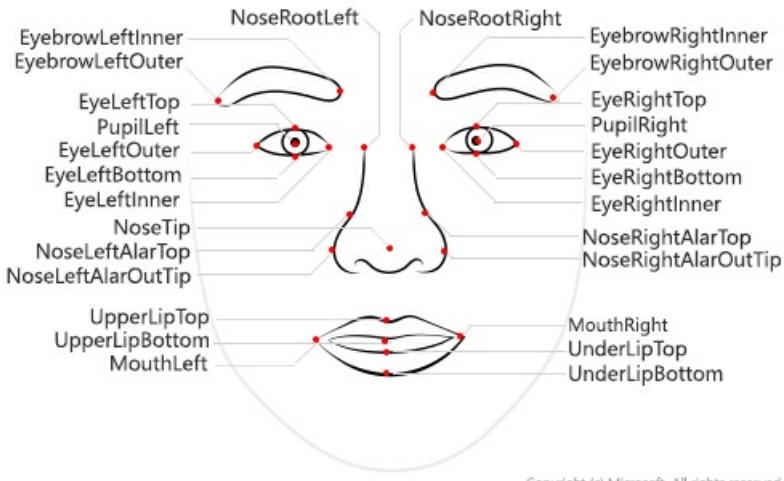
foreach (var face in faces)
{
    var rect = face.FaceRectangle;
    var landmarks = face.FaceLandmarks;
}
```

The FaceRectangle property that is returned with detected faces is essentially locations on the face in pixels. Usually, this rectangle contains the eyes, eyebrows, the nose and the mouth –the top of head, ears and the chin are not included. If you crop a complete head or mid-shot portrait (a photo ID type image), you may want to expand the area of the rectangular face frame because the area of the face may be too small for some applications. To locate a face more precisely, using face landmarks (locate face features or face direction mechanisms) described in the next section will prove to be very useful.

## Step 3: Understanding and using face landmarks

Face landmarks are a series of specifically detailed points on a face; typically points of face components like the pupils, canthus or nose. Face landmarks are optional attributes that can be analyzed during face detection. You can either pass 'true' as a Boolean value to the returnFaceLandmarks query parameter when calling the [Face - Detect](#), or use the returnFaceLandmarks optional parameter for the FaceServiceClient class DetectAsync method in order to include the face landmarks in the detection results.

By default, there are 27 predefined landmark points. The following figure shows how all 27 points are defined:



Copyright (c) Microsoft. All rights reserved.

The points returned are in units of pixels, just like the face rectangular frame. Therefore making it easier to mark specific points of interest in the image. The following code demonstrates retrieving the locations of the nose and pupils:

```
var faces = await faceServiceClient.DetectAsync(imageUrl, returnFaceLandmarks:true);

foreach (var face in faces)
{
    var rect = face.FaceRectangle;
    var landmarks = face.FaceLandmarks;

    double noseX = landmarks.NoseTip.X;
    double noseY = landmarks.NoseTip.Y;

    double leftPupilX = landmarks.PupilLeft.X;
    double leftPupilY = landmarks.PupilLeft.Y;

    double rightPupilX = landmarks.PupilRight.X;
    double rightPupilY = landmarks.PupilRight.Y;
}
```

In addition to marking face features in an image, face landmarks can also be used to accurately calculate the direction of the face. For example, we can define the direction of the face as a vector from the center of the mouth to the center of the eyes. The code below explains this in detail:

```
var landmarks = face.FaceLandmarks;

var upperLipBottom = landmarks.UpperLipBottom;
var underLipTop = landmarks.UnderLipTop;

var centerOfMouth = new Point(
    (upperLipBottom.X + underLipTop.X) / 2,
    (upperLipBottom.Y + underLipTop.Y) / 2);

var eyeLeftInner = landmarks.EyeLeftInner;
var eyeRightInner = landmarks.EyeRightInner;

var centerOfTwoEyes = new Point(
    (eyeLeftInner.X + eyeRightInner.X) / 2,
    (eyeLeftInner.Y + eyeRightInner.Y) / 2);

Vector faceDirection = new Vector(
    centerOfTwoEyes.X - centerOfMouth.X,
    centerOfTwoEyes.Y - centerOfMouth.Y);
```

By knowing the direction that the face is in, you can then rotate the rectangular face frame to align it with the face. It is clear that using face landmarks can provide more detail and utility.

## Step 4: Using other face attributes

Besides face landmarks, Face – Detect API can also analyze several other attributes on a face. These attributes include:

- Age
- Gender
- Smile intensity
- Facial hair
- A 3D head pose

These attributes are predicted by using statistical algorithms and may not always be 100% precise. However, they are still helpful when you want to classify faces by these attributes. For more information about each of the attributes, please refer to the [Glossary](#).

Below is a simple example of extracting face attributes during face detection:

```
var requiredFaceAttributes = new FaceAttributeType[] {
    FaceAttributeType.Age,
    FaceAttributeType.Gender,
    FaceAttributeType.Smile,
    FaceAttributeType.FacialHair,
    FaceAttributeType.HeadPose,
    FaceAttributeType.Glasses
};

var faces = await faceServiceClient.DetectAsync(imageUrl,
    returnFaceLandmarks: true,
    returnFaceAttributes: requiredFaceAttributes);

foreach (var face in faces)
{
    var id = face.FaceId;
    var attributes = face.FaceAttributes;
    var age = attributes.Age;
    var gender = attributes.Gender;
    var smile = attributes.Smile;
    var facialHair = attributes.FacialHair;
    var headPose = attributes.HeadPose;
    var glasses = attributes.Glasses;
}
```

## Summary

In this guide you have learned the functionalities of [Face - Detect](#) API, how it can detect faces for local uploaded images or image URLs on the web; how it can detect faces by returning rectangular face frames; and how it can also analyze face landmarks, 3D head poses and other face attributes as well.

For more information about API details, please refer to the API reference guide for [Face - Detect](#).

## Related Topics

[How to Identify Faces in Image](#)

# How to Identify Faces in Image

4/12/2017 • 6 min to read • [Edit Online](#)

This guide will demonstrate how to identify unknown faces using person groups, which are created from known people in advance. The samples are written in C# using the Face API client library.

## Concepts

If you are not familiar with the following concepts in this guide, please search for the definitions in our [glossary](#) at any time:

- Face - Detect
- Face - Identify
- Person Group

## Preparation

In this sample, we will demonstrate the following:

- How to create a person group - This person group contains a list of known people.
- How to assign faces to each person - These faces are used as a baseline to identify people. It is recommended to use clear front faces, just like your photo ID. A good set of photos should contain faces of the same person, yet have different poses, clothes colors or hair styles.

To carry out the demonstration of this sample, you will need to prepare a bunch of pictures:

- A few photos with the person's face. [Click here to download sample photos](#) for Anna, Bill and Clare.
- A series of test photos, which may or may not contain the faces of Anna, Bill or Clare used to test their identification. You can also select some sample images from the link above.

## Step 1: Authorize the API call

Every call to the Face API requires a subscription key. This key needs to be either passed through a query string parameter, or specified in the request header. To pass the subscription key through query string, please refer to the request URL for the [Face - Detect](#) as an example:

```
https://westus.api.cognitive.microsoft.com/face/v1.0/detect[?returnFaceId][&returnFaceLandmarks]  
[&returnFaceAttributes]  
&subscription-key=<Your subscription key>
```

As an alternative, the subscription key can also be specified in the HTTP request header: **ocp-apim-subscription-key: <Your subscription key>** When using a client library, the subscription key is passed in through the constructor of the FaceServiceClient class. For example:

```
faceServiceClient = new FaceServiceClient("Your subscription key");
```

The subscription key can be obtained from the Marketplace page of your Azure management portal. See [Subscriptions](#).

## Step 2: Create the person group

In this step, we created a person group named "MyFriends" that contains three people: Anna, Bill and Clare. Each person will have several faces registered. The faces need to be detected from the images. After all of these steps, you will have a person group like the image below:



### 2.1 Define people for the person group

A person is a basic unit of identify. A person can have one or more known faces registered. However, a person group is a collection of people, and each person is defined within a particular person group. The identify is done against a person group. So, the task is to create a person group, and then create people in it, such as Anna, Bill, and Clare.

First, you need to create a new person group. This is executed by using the [Person Group - Create a Person Group](#) API. The corresponding client library API is the CreatePersonGroupAsync method for the FaceServiceClient class. The group ID specified to create the group is unique for each subscription –you can also get, update or delete person groups using other person group APIs. Once a group is defined, people can then be defined within it by using the [Person - Create a Person](#) API. The client library method is CreatePersonAsync. You can add face to each person after they're created.

```
// Create an empty person group
string personGroupId = "myfriends";
await faceServiceClient.CreatePersonGroupAsync(personGroupId, "My Friends");

// Define Anna
CreatePersonResult friend1 = await faceServiceClient.CreatePersonAsync(
    // Id of the person group that the person belonged to
    personGroupId,
    // Name of the person
    "Anna"
);
// Define Bill and Clare in the same way
```

### 2.2 Detect faces and register them to correct person

Detection is done by sending a "POST" web request to the [Face - Detect](#) API with the image file in the HTTP request body. When using the client library, face detection is executed through the DetectAsync method for the FaceServiceClient class.

For each face detected you can call [Person – Add a Person Face](#) to add it to the correct person.

The following code below demonstrates the process of how to detect a face from an image and add it to a person:

```
// Directory contains image files of Anna
const string friend1ImageDir = @"D:\Pictures\MyFriends\Anna\";

foreach (string imagePath in Directory.GetFiles(friend1ImageDir, "*.jpg"))
{
    using (Stream s = File.OpenRead(imagePath))
    {
        // Detect faces in the image and add to Anna
        await faceServiceClient.AddPersonFaceAsync(
            personGroupId, friend1.PersonId, s);
    }
}
// Do the same for Bill and Clare
```

Notice that if the image contains more than one face, only the largest face will be added. You can add other faces to the person by passing a string in the format of "targetFace = left, top, width, height" to [Person – Add a Person Face](#) API's targetFace query parameter, or using the targetFace optional parameter for the AddPersonFaceAsync method to add other faces. Each face added to the person will be given a unique persisted face ID, which can be used in [Person – Delete a Person Face](#) and [Face – Identify](#).

## Step 3: Train the person group

The person group must be trained before an identification can be performed using it. Moreover, it has to be re-trained after adding or removing any person, or if any person has their registered face edited. The training is done by the [Person Group – Train Person Group](#) API. When using the client library, it is simply a call to the TrainPersonGroupAsync method:

```
await faceServiceClient.TrainPersonGroupAsync(personGroupId);
```

Please note that the training is an asynchronous process. It may not be finished even after the the TrainPersonGroupAsync method returned. You may need to query the training status by [Person Group - Get Person Group Training Status](#) API or GetPersonGroupTrainingStatusAsync method of the client library. The following code demonstrates a simple logic of waiting person group training to finish:

```
TrainingStatus trainingStatus = null;
while(true)
{
    trainingStatus = await faceServiceClient.GetPersonGroupTrainingStatusAsync(personGroupId);

    if (trainingStatus.Status != "running")
    {
        break;
    }

    await Task.Delay(1000);
}
```

## Step 4: Identify a face against a defined person group

When performing identify, the Face API can compute the similarity of a test face among all the faces within a group, and returns the most comparable person(s) for that testing face. This is done through the [Face - Identify](#) API or the IdentifyAsync method of the client library.

The testing face needs to be detected using the previous steps listed above, and then the face ID is passed to the identify API as a second argument. Multiple face IDs can be identified at once, and the result will contain all the identify results. By default, the identify returns only one person which matches the test face best. If you prefer, you can specify the optional parameter `maxNumOfCandidatesReturned` to let the identify return more candidates.

The following code below demonstrates the process of identify:

```
string testImageFile = @"D:\Pictures\test_img1.jpg";

using (Stream s = File.OpenRead(testImageFile))
{
    var faces = await faceServiceClient.DetectAsync(s);
    var faceIds = faces.Select(face => face.FaceId).ToArray();

    var results = await faceServiceClient.IdentifyAsync(personGroupId, faceIds);
    foreach (var identifyResult in results)
    {
        Console.WriteLine("Result of face: {0}", identifyResult.FaceId);
        if (identifyResult.Candidates.Length == 0)
        {
            Console.WriteLine("No one identified");
        }
        else
        {
            // Get top 1 among all candidates returned
            var candidateId = identifyResult.Candidates[0].PersonId;
            var person = await faceServiceClient.GetPersonAsync(personGroupId, candidateId);
            Console.WriteLine("Identified as {0}", person.Name);
        }
    }
}
```

When you have finished the steps, you can try to identify different faces and see if the faces of Anna, Bill or Clare can be correctly identified, according to the image(s) uploaded for face detection. See the examples below :



## Summary

In this guide you have learned the process of creating a person group and identifying a person. The following are a quick reminder of the features previously explained and demonstrated:

- Detecting faces using the [Face - Detect API](#)
- Creating person groups using the [Person Group - Create a Person Group API](#)
- Creating persons using the [Person - Create a Person API](#)
- Train a person group using the [Person Group – Train Person Group API](#)

- Identifying unknown faces against the person group using the [Face - Identify](#) API

## Related Topics

[How to Detect Faces in Image](#)

# How to Analyze Videos in Real-time

4/12/2017 • 7 min 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 uses 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();

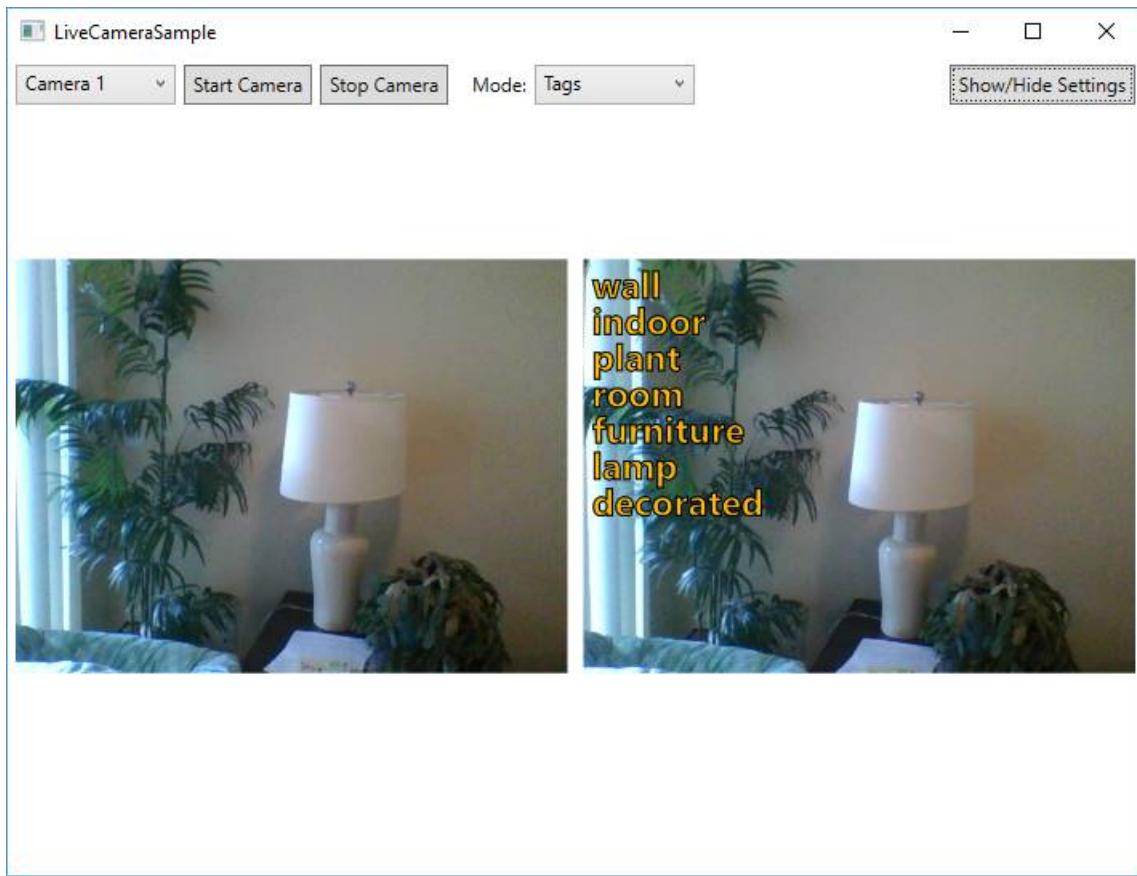
            // Wait for keypress to stop
            Console.WriteLine("Press any key to stop...");
            Console.ReadKey();

            // 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 [microsoft.com/cognitive](https://microsoft.com/cognitive). 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.

To report abuse of the Microsoft Cognitive Services to Microsoft, please visit the [Microsoft Cognitive Services website](#), and use the "Report Abuse" link at the bottom of the page to contact Microsoft. For more information about Microsoft privacy policies please see their privacy statement here: <https://go.microsoft.com/fwlink/?LinkId=521839>.

## 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](#).

## Related Topics

- [How to Identify Faces in Image](#)
- [How to Detect Faces in Image](#)

# Face API cURL Quick Starts

4/18/2017 • 2 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with cURL to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Identify Faces in Images](#)

Learn more about obtaining free Subscription Keys [here](#).

## Detect Faces in Images With Face API Using cURL

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

### Face Detect cURL Example Request

```
@ECHO OFF

curl -v -X POST "https://westus.api.cognitive.microsoft.com/face/v1.0/detect?
returnFaceId=true&returnFaceLandmarks=false&returnFaceAttributes={string}"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"
```

### Face - Detect Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[

{
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "faceRectangle": {
        "width": 78,
        "height": 78,
        "left": 394,
        "top": 54
    },
    "faceLandmarks": {
        "pupilLeft": {
            "x": 412.7,
            "y": 78.4
        },
        "pupilRight": {
            "x": 446.8,
            "y": 74.2
        },
        "noseTip": {
            "x": 437.7,
            "y": 92.4
        },
        "mouthLeft": {
```

```
        "x": 417.8,
        "y": 114.4
    },
    "mouthRight": {
        "x": 451.3,
        "y": 109.3
    },
    "eyebrowLeftOuter": {
        "x": 397.9,
        "y": 78.5
    },
    "eyebrowLeftInner": {
        "x": 425.4,
        "y": 70.5
    },
    "eyeLeftOuter": {
        "x": 406.7,
        "y": 80.6
    },
    "eyeLeftTop": {
        "x": 412.2,
        "y": 76.2
    },
    "eyeLeftBottom": {
        "x": 413.0,
        "y": 80.1
    },
    "eyeLeftInner": {
        "x": 418.9,
        "y": 78.0
    },
    "eyebrowRightInner": {
        "x": 4.8,
        "y": 69.7
    },
    "eyebrowRightOuter": {
        "x": 5.5,
        "y": 68.5
    },
    "eyeRightInner": {
        "x": 441.5,
        "y": 75.0
    },
    "eyeRightTop": {
        "x": 446.4,
        "y": 71.7
    },
    "eyeRightBottom": {
        "x": 447.0,
        "y": 75.3
    },
    "eyeRightOuter": {
        "x": 451.7,
        "y": 73.4
    },
    "noseRootLeft": {
        "x": 428.0,
        "y": 77.1
    },
    "noseRootRight": {
        "x": 435.8,
        "y": 75.6
    },
    "noseLeftAlarTop": {
        "x": 428.3,
        "y": 89.7
    },
    "noseRightAlarTop": {
        "x": 442.2,
```

```

        "y": 87.0
    },
    "noseLeftAlarOutTip": {
        "x": 424.3,
        "y": 96.4
    },
    "noseRightAlarOutTip": {
        "x": 446.6,
        "y": 92.5
    },
    "upperLipTop": {
        "x": 437.6,
        "y": 105.9
    },
    "upperLipBottom": {
        "x": 437.6,
        "y": 108.2
    },
    "underLipTop": {
        "x": 436.8,
        "y": 111.4
    },
    "underLipBottom": {
        "x": 437.3,
        "y": 114.5
    }
},
"faceAttributes": {
    "age": 71.0,
    "gender": "male",
    "smile": 0.88,
    "facialHair": {
        "mustache": 0.8,
        "beard": 0.1,
        "sideburns": 0.02
    },
    "glasses": "sunglasses",
    "headPose": {
        "roll": 2.1,
        "yaw": 3,
        "pitch": 0
    }
}
}
]

```

## Identify Faces in Images With Face API Using cURL

Use the [Face - Identify method](#) identify people based on a detected face and people database (defined as a person group) which needs to be created in advance and can be edited over time

### Face - Identify cURL Example Request

```

@ECHO OFF

curl -v -X POST "https://westus.api.cognitive.microsoft.com/face/v1.0/identify"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"

```

### Face - Identify Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[  
  {  
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",  
    "candidates": [  
      {  
        "personId": "25985303-c537-4467-b41d-bdb45cd95ca1",  
        "confidence": 0.92  
      }  
    ]  
  },  
  {  
    "faceId": "65d083d4-9447-47d1-af30-b626144bf0fb",  
    "candidates": [  
      {  
        "personId": "2ae4935b-9659-44c3-977f-61fac20d0538",  
        "confidence": 0.89  
      }  
    ]  
  }  
]
```

# Face API C# Quick Starts

4/18/2017 • 3 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with C# to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Create a Person Group](#)

## Prerequisites

- Get the Microsoft Face API Windows SDK [here](#)
- Learn more about obtaining free Subscription Keys [here](#)

## Detect Faces in Images With Face API Using C#

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

### Face Detect C# Example Request

The sample is written in C# using the Face API client library.

```

using System;
using System.IO;
using System.Net.Http.Headers;
using System.Net.Http;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter the path to the JPEG image with faces to identify:");
            string imagePath = Console.ReadLine();

            MakeDetectRequest(imagePath);

            Console.WriteLine("\n\n\nWait for the result below, then hit ENTER to exit...\n\n\n");
            Console.ReadLine();
        }

        static byte[] GetImageAsByteArray(string imagePath)
        {
            FileStream fileStream = new FileStream(imagePath, FileMode.Open, FileAccess.Read);
            BinaryReader binaryReader = new BinaryReader(fileStream);
            return binaryReader.ReadBytes((int)fileStream.Length);
        }

        static async void MakeDetectRequest(string imagePath)
        {
            var client = new HttpClient();

            // Request headers - replace this example key with your valid key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "6726adbabb494773a28a7a5a21d5974a");

            // Request parameters and URI string.
            string queryString = "returnFaceId=true&returnFaceLandmarks=false&returnFaceAttributes=age,gender";
            string uri = "https://westus.api.cognitive.microsoft.com/face/v1.0/detect?" + queryString;

            HttpResponseMessage response;
            string responseContent;

            // Request body. Try this sample with a locally stored JPEG image.
            byte[] byteData = GetImageAsByteArray(imagePath);

            using (var 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");
                response = await client.PostAsync(uri, content);
                responseContent = response.Content.ReadAsStringAsync().Result;
            }

            //A peek at the JSON response.
            Console.WriteLine(responseContent);
        }
    }
}

```

## Face - Detect Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
  {
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "faceRectangle": {
      "top": 100,
      "left": 100,
      "width": 100,
      "height": 100
    },
    "faceAttributes": {
      "age": 30,
      "gender": "Male"
    }
  }
]
```

```
    "width": 78,
    "height": 78,
    "left": 394,
    "top": 54
},
"faceLandmarks": {
    "pupilLeft": {
        "x": 412.7,
        "y": 78.4
    },
    "pupilRight": {
        "x": 446.8,
        "y": 74.2
    },
    "noseTip": {
        "x": 437.7,
        "y": 92.4
    },
    "mouthLeft": {
        "x": 417.8,
        "y": 114.4
    },
    "mouthRight": {
        "x": 451.3,
        "y": 109.3
    },
    "eyebrowLeftOuter": {
        "x": 397.9,
        "y": 78.5
    },
    "eyebrowLeftInner": {
        "x": 425.4,
        "y": 70.5
    },
    "eyeLeftOuter": {
        "x": 406.7,
        "y": 80.6
    },
    "eyeLeftTop": {
        "x": 412.2,
        "y": 76.2
    },
    "eyeLeftBottom": {
        "x": 413.0,
        "y": 80.1
    },
    "eyeLeftInner": {
        "x": 418.9,
        "y": 78.0
    },
    "eyebrowRightInner": {
        "x": 4.8,
        "y": 69.7
    },
    "eyebrowRightOuter": {
        "x": 5.5,
        "y": 68.5
    },
    "eyeRightInner": {
        "x": 441.5,
        "y": 75.0
    },
    "eyeRightTop": {
        "x": 446.4,
        "y": 71.7
    },
    "eyeRightBottom": {
        "x": 447.0,
        "y": 75.3
    }
}
```

```

        },
        "eyeRightOuter": {
            "x": 451.7,
            "y": 73.4
        },
        "noseRootLeft": {
            "x": 428.0,
            "y": 77.1
        },
        "noseRootRight": {
            "x": 435.8,
            "y": 75.6
        },
        "noseLeftAlarTop": {
            "x": 428.3,
            "y": 89.7
        },
        "noseRightAlarTop": {
            "x": 442.2,
            "y": 87.0
        },
        "noseLeftAlarOutTip": {
            "x": 424.3,
            "y": 96.4
        },
        "noseRightAlarOutTip": {
            "x": 446.6,
            "y": 92.5
        },
        "upperLipTop": {
            "x": 437.6,
            "y": 105.9
        },
        "upperLipBottom": {
            "x": 437.6,
            "y": 108.2
        },
        "underLipTop": {
            "x": 436.8,
            "y": 111.4
        },
        "underLipBottom": {
            "x": 437.3,
            "y": 114.5
        }
    },
    "faceAttributes": {
        "age": 71.0,
        "gender": "male",
        "smile": 0.88,
        "facialHair": {
            "mustache": 0.8,
            "beard": 0.1,
            "sideburns": 0.02
        },
        "glasses": "sunglasses",
        "headPose": {
            "roll": 2.1,
            "yaw": 3,
            "pitch": 0
        }
    }
}
]

```

## Create a Person Group With Face API Using C#

Use the [Person Group - Create a Person Group method](#) to create a new person group with specified personGroupId, name, and user-provided userData.

#### Person Group - Create a Person Group C# Example Request

```
using System;
using System.Net.Http.Headers;
using System.Net.Http;

namespace CSHttpClientSample
{
    static class Program
    {
        static void Main()
        {
            Console.WriteLine("Enter an ID for the group you wish to create:");
            Console.WriteLine("(Use numbers, lower case letters, '-' and '_'. The maximum length of the personGroupId is 64.)");

            string personGroupId = Console.ReadLine();
            MakeCreateGroupRequest(personGroupId);

            Console.WriteLine("\n\n\nWait for the result below, then hit ENTER to exit...\n\n\n");
            Console.ReadLine();
        }

        static async void MakeCreateGroupRequest(string personGroupId)
        {
            var client = new HttpClient();

            // Request headers - replace this example key with your valid key.
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "6726adbabb494773a28a7a5a21d5974a");

            // Request URI string.
            string uri = "https://westus.api.cognitive.microsoft.com/face/v1.0/persongroups/" + personGroupId;

            // Here "name" is for display and doesn't have to be unique. Also, "userData" is optional.
            string json = "{\"name\":\"My Group\", \"userData\":\"Some data related to my group.\\"}";
            HttpContent content = new StringContent(json);
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

            HttpResponseMessage response = await client.PutAsync(uri, content);

            // If the group was created successfully, you'll see "OK".
            // Otherwise, if a group with the same personGroupId has been created before, you'll see "Conflict".
            Console.WriteLine("Response status: " + response.StatusCode);
        }
    }
}
```

# Face API Java Quick Starts

4/18/2017 • 3 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with Java to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Create a Person Group](#)

## Prerequisites

- Get the Microsoft Face API Android SDK [here](#)
- Learn more about obtaining free subscription keys [here](#)

## Detect Faces in Images with Face API Using Java

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

### **Face Detect Java Example Request**

```

// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
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.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            URIBuilder uriBuilder = new
URIBuilder("https://westus.api.cognitive.microsoft.com/face/v1.0/detect");

            uriBuilder.setParameter("returnFaceId", "true");
            uriBuilder.setParameter("returnFaceLandmarks", "false");
            uriBuilder.setParameter("returnFaceAttributes", "age");

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

            // Request headers. Replace the example key below with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. Replace the example URL below with the URL of the image you want to analyze.
            StringEntity reqEntity = new StringEntity("{\"url\":\"http://example.com/1.jpg\"}");
            request.setEntity(reqEntity);

            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                System.out.println(EntityUtils.toString(entity));
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

#### Face - Detect Response

A successful response will be returned in JSON. The following is an example of a successful response:

```
[
  {
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "faceRectangle": {
      "width": 78,
      "height": 78,
      "left": 394,
      "top": 54
    },
    "faceLandmarks": {

```

```
"pupilLeft": {  
    "x": 412.7,  
    "y": 78.4  
},  
"pupilRight": {  
    "x": 446.8,  
    "y": 74.2  
},  
"noseTip": {  
    "x": 437.7,  
    "y": 92.4  
},  
"mouthLeft": {  
    "x": 417.8,  
    "y": 114.4  
},  
"mouthRight": {  
    "x": 451.3,  
    "y": 109.3  
},  
"eyebrowLeftOuter": {  
    "x": 397.9,  
    "y": 78.5  
},  
"eyebrowLeftInner": {  
    "x": 425.4,  
    "y": 70.5  
},  
"eyeLeftOuter": {  
    "x": 406.7,  
    "y": 80.6  
},  
"eyeLeftTop": {  
    "x": 412.2,  
    "y": 76.2  
},  
"eyeLeftBottom": {  
    "x": 413.0,  
    "y": 80.1  
},  
"eyeLeftInner": {  
    "x": 418.9,  
    "y": 78.0  
},  
"eyebrowRightInner": {  
    "x": 4.8,  
    "y": 69.7  
},  
"eyebrowRightOuter": {  
    "x": 5.5,  
    "y": 68.5  
},  
"eyeRightInner": {  
    "x": 441.5,  
    "y": 75.0  
},  
"eyeRightTop": {  
    "x": 446.4,  
    "y": 71.7  
},  
"eyeRightBottom": {  
    "x": 447.0,  
    "y": 75.3  
},  
"eyeRightOuter": {  
    "x": 451.7,  
    "y": 73.4  
},  
"noseRootLeft": {
```

```

        "x": 428.0,
        "y": 77.1
    },
    "noseRootRight": {
        "x": 435.8,
        "y": 75.6
    },
    "noseLeftAlarTop": {
        "x": 428.3,
        "y": 89.7
    },
    "noseRightAlarTop": {
        "x": 442.2,
        "y": 87.0
    },
    "noseLeftAlarOutTip": {
        "x": 424.3,
        "y": 96.4
    },
    "noseRightAlarOutTip": {
        "x": 446.6,
        "y": 92.5
    },
    "upperLipTop": {
        "x": 437.6,
        "y": 105.9
    },
    "upperLipBottom": {
        "x": 437.6,
        "y": 108.2
    },
    "underLipTop": {
        "x": 436.8,
        "y": 111.4
    },
    "underLipBottom": {
        "x": 437.3,
        "y": 114.5
    }
},
"faceAttributes": {
    "age": 71.0,
    "gender": "male",
    "smile": 0.88,
    "facialHair": {
        "mustache": 0.8,
        "beard": 0.1,
        "sideburns": 0.02
    },
    "glasses": "sunglasses",
    "headPose": {
        "roll": 2.1,
        "yaw": 3,
        "pitch": 0
    }
}
]

```

## Create a Person Group with Face API Using Java

Use the [Person Group - Create a Person Group method](#) to create a new person group with specified personGroupId, name, and user-provided userData. A person group is one of the most important parameters for the Face - Identify API. The Identify API searches for persons' faces in a specified person group.

### Person Group - Create a Person Group Example

```

// // This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class Main
{
    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            // The valid characters for the ID below include numbers, English letters in lower case, '-', and
            '_'.

            // The maximum length of the personGroupId is 64.
            String personGroupId = "example-group-00";

            URIBuilder builder = new
URIBuilder("https://westus.api.cognitive.microsoft.com/face/v1.0/persongroups/" +
personGroupId);

            URI uri = builder.build();
            HttpPut request = new HttpPut(uri);

            // Request headers. Replace the example key with your valid subscription key.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", "13hc77781f7e4b19b5fcdd72a8df7156");

            // Request body. The name field is the display name you want for the group (must be under 128
characters).
            // The size limit for what you want to include in the userData field is 16KB.
            String body = "{ \"name\":\"My Group\", \"userData\":\"User-provided data attached to the person
group.\" }";

            StringEntity reqEntity = new StringEntity(body);
            request.setEntity(reqEntity);

            HttpResponse response = httpClient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                System.out.println(EntityUtils.toString(entity));
            }
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

```

# Face API JavaScript Quick Starts

4/18/2017 • 2 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with JavaScript to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Identify Faces in Images](#)

Learn more about obtaining free Subscription Keys [here](#)

## Detect Faces in Images With Face API Using JavaScript

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

**Face Detect JavaScript Example Request**

```

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

<script type="text/javascript">
$(function() {
    var params = {
        // Request parameters
        "returnFaceId": "true",
        "returnFaceLandmarks": "false",
        "returnFaceAttributes": "{string}",
    };

    $.ajax({
        url: "https://westus.api.cognitive.microsoft.com/face/v1.0/detect?" + $.param(params),
        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","{subscription key}");
        },
        type: "POST",
        // Request body
        data: "{body}",
    })
    .done(function(data) {
        alert("success");
    })
    .fail(function() {
        alert("error");
    });
});
</script>
</body>
</html>

```

#### Face - Detect Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
{
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "faceRectangle": {
        "width": 78,
        "height": 78,
        "left": 394,
        "top": 54
    },
    "faceLandmarks": {
        "pupilLeft": {
            "x": 412.7,
            "y": 78.4
        },
        "pupilRight": {
            "x": 446.8,
            "y": 74.2
        },
        "noseTip": {
            "x": 437.7,
            "y": 92.4
        },
        "mouthLeft": {
            "x": 417.8,
            "y": 102.4
        }
    }
}
```

```
        "y": 114.4
    },
    "mouthRight": {
        "x": 451.3,
        "y": 109.3
    },
    "eyebrowLeftOuter": {
        "x": 397.9,
        "y": 78.5
    },
    "eyebrowLeftInner": {
        "x": 425.4,
        "y": 70.5
    },
    "eyeLeftOuter": {
        "x": 406.7,
        "y": 80.6
    },
    "eyeLeftTop": {
        "x": 412.2,
        "y": 76.2
    },
    "eyeLeftBottom": {
        "x": 413.0,
        "y": 80.1
    },
    "eyeLeftInner": {
        "x": 418.9,
        "y": 78.0
    },
    "eyebrowRightInner": {
        "x": 4.8,
        "y": 69.7
    },
    "eyebrowRightOuter": {
        "x": 5.5,
        "y": 68.5
    },
    "eyeRightInner": {
        "x": 441.5,
        "y": 75.0
    },
    "eyeRightTop": {
        "x": 446.4,
        "y": 71.7
    },
    "eyeRightBottom": {
        "x": 447.0,
        "y": 75.3
    },
    "eyeRightOuter": {
        "x": 451.7,
        "y": 73.4
    },
    "noseRootLeft": {
        "x": 428.0,
        "y": 77.1
    },
    "noseRootRight": {
        "x": 435.8,
        "y": 75.6
    },
    "noseLeftAlarTop": {
        "x": 428.3,
        "y": 89.7
    },
    "noseRightAlarTop": {
        "x": 442.2,
        "y": 87.0
    }
}
```

```

        },
        "noseLeftAlarOutTip": {
            "x": 424.3,
            "y": 96.4
        },
        "noseRightAlarOutTip": {
            "x": 446.6,
            "y": 92.5
        },
        "upperLipTop": {
            "x": 437.6,
            "y": 105.9
        },
        "upperLipBottom": {
            "x": 437.6,
            "y": 108.2
        },
        "underLipTop": {
            "x": 436.8,
            "y": 111.4
        },
        "underLipBottom": {
            "x": 437.3,
            "y": 114.5
        }
    },
    "faceAttributes": {
        "age": 71.0,
        "gender": "male",
        "smile": 0.88,
        "facialHair": {
            "mustache": 0.8,
            "beard": 0.1,
            "sideburns": 0.02
        },
        "glasses": "sunglasses",
        "headPose": {
            "roll": 2.1,
            "yaw": 3,
            "pitch": 0
        }
    }
}
]

```

## Identify Faces in Images With Face API Using JavaScript

Use the [Face - Identify method](#) identify people based on a detected face and people database (defined as a person group) which needs to be created in advance and can be edited over time

### [Face - Identify JavaScript Example Request](#)

```

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

<script type="text/javascript">
$(function() {
    var params = {
        // Request parameters
    };

    $.ajax({
        url: "https://westus.api.cognitive.microsoft.com/face/v1.0/identify?" + $.param(params),
        beforeSend: function(xhrObj){
            // Request headers
            xhrObj.setRequestHeader("Content-Type","application/json");
            xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","{subscription key}");
        },
        type: "POST",
        // Request body
        data: "{body}",
    })
    .done(function(data) {
        alert("success");
    })
    .fail(function() {
        alert("error");
    });
});
});
</script>
</body>
</html>

```

### Face - Identify Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
  {
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "candidates": [
      {
        "personId": "25985303-c537-4467-b41d-bdb45cd95ca1",
        "confidence": 0.92
      }
    ]
  },
  {
    "faceId": "65d083d4-9447-47d1-af30-b626144bf0fb",
    "candidates": [
      {
        "personId": "2ae4935b-9659-44c3-977f-61fac20d0538",
        "confidence": 0.89
      }
    ]
  }
]
```

# Face API PHP Quick Starts

4/18/2017 • 3 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with PHP to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Identify Faces in Images](#)

Learn more about obtaining free Subscription Keys [here](#).

## Detect Faces in Images With Face API Using PHP

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

### **Face Detect PHP Example Request**

```

<?php

// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)

require_once 'HTTP/Request2.php';

$request = new Http_Request2('https://westus.api.cognitive.microsoft.com/face/v1.0/detect');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',
    'Ocp-Apim-Subscription-Key' => '{subscription key}',
);

$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'returnFaceId' => 'true',
    'returnFaceLandmarks' => 'false',
    'returnFaceAttributes' => '{string}',
);

$url->setQueryVariables($parameters);

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

// Request body
$request->setBody("{body}");

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

#### Face - Detect Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
{
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "faceRectangle": {
        "width": 78,
        "height": 78,
        "left": 394,
        "top": 54
    },
    "faceLandmarks": {
        "pupilLeft": {
            "x": 412.7,
            "y": 78.4
        },
        "pupilRight": {
            "x": 446.8,
            "y": 74.2
        },
        "noseTip": {
            "x": 437.7,
            "y": 92.4
        }
    }
}
```

```
},
"mouthLeft": {
    "x": 417.8,
    "y": 114.4
},
"mouthRight": {
    "x": 451.3,
    "y": 109.3
},
"eyebrowLeftOuter": {
    "x": 397.9,
    "y": 78.5
},
"eyebrowLeftInner": {
    "x": 425.4,
    "y": 70.5
},
"eyeLeftOuter": {
    "x": 406.7,
    "y": 80.6
},
"eyeLeftTop": {
    "x": 412.2,
    "y": 76.2
},
"eyeLeftBottom": {
    "x": 413.0,
    "y": 80.1
},
"eyeLeftInner": {
    "x": 418.9,
    "y": 78.0
},
"eyebrowRightInner": {
    "x": 4.8,
    "y": 69.7
},
"eyebrowRightOuter": {
    "x": 5.5,
    "y": 68.5
},
"eyeRightInner": {
    "x": 441.5,
    "y": 75.0
},
"eyeRightTop": {
    "x": 446.4,
    "y": 71.7
},
"eyeRightBottom": {
    "x": 447.0,
    "y": 75.3
},
"eyeRightOuter": {
    "x": 451.7,
    "y": 73.4
},
"noseRootLeft": {
    "x": 428.0,
    "y": 77.1
},
"noseRootRight": {
    "x": 435.8,
    "y": 75.6
},
"noseLeftAlarTop": {
    "x": 428.3,
    "y": 89.7
}.
```

```

        },
        "noseRightAlarTop": {
            "x": 442.2,
            "y": 87.0
        },
        "noseLeftAlarOutTip": {
            "x": 424.3,
            "y": 96.4
        },
        "noseRightAlarOutTip": {
            "x": 446.6,
            "y": 92.5
        },
        "upperLipTop": {
            "x": 437.6,
            "y": 105.9
        },
        "upperLipBottom": {
            "x": 437.6,
            "y": 108.2
        },
        "underLipTop": {
            "x": 436.8,
            "y": 111.4
        },
        "underLipBottom": {
            "x": 437.3,
            "y": 114.5
        }
    },
    "faceAttributes": {
        "age": 71.0,
        "gender": "male",
        "smile": 0.88,
        "facialHair": {
            "mustache": 0.8,
            "beard": 0.1,
            "sideburns": 0.02
        },
        "glasses": "sunglasses",
        "headPose": {
            "roll": 2.1,
            "yaw": 3,
            "pitch": 0
        }
    }
}
]

```

## Identify Faces in Images With Face API Using PHP

Use the [Face - Identify method](#) identify people based on a detected face and people database (defined as a person group) which needs to be created in advance and can be edited over time

### **Face - Identify PHP Example Request**

```

<?php
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-
ga/)
require_once 'HTTP/Request2.php';

$request = new Http_Request2('https://westus.api.cognitive.microsoft.com/face/v1.0/identify');
$url = $request->getUrl();

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

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

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

// Request body
$request->setBody("{body}");

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

### Face - Identify Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
  {
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "candidates": [
      {
        "personId": "25985303-c537-4467-b41d-bdb45cd95ca1",
        "confidence": 0.92
      }
    ]
  },
  {
    "faceId": "65d083d4-9447-47d1-af30-b626144bf0fb",
    "candidates": [
      {
        "personId": "2ae4935b-9659-44c3-977f-61fac20d0538",
        "confidence": 0.89
      }
    ]
  }
]
```

# Face API Python Quick Starts

4/18/2017 • 4 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with Python to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Create a Person Group](#)

Learn more about obtaining free Subscription Keys [here](#)

## Detect Faces in Images With Face API Using Python

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

### **Face Detect Python Example Request**

```

##### Python 2.7 #####
import httplib, urllib, base64

headers = {
    # Request headers
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '{subscription key}',
}

params = urllib.urlencode({
    # Request parameters
    'returnFaceId': 'true',
    'returnFaceLandmarks': 'false',
    'returnFaceAttributes': '{string}',
})

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/face/v1.0/detect?%s" % params, "{body}", headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####
##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64

headers = {
    # Request headers
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '{subscription key}',
}

params = urllib.parse.urlencode({
    # Request parameters
    'returnFaceId': 'true',
    'returnFaceLandmarks': 'false',
    'returnFaceAttributes': '{string}',
})

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/face/v1.0/detect?%s" % params, "{body}", headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####

```

### Face - Detect Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
{
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "faceRectangle": {
        "width": 78,
        "height": 78,
        "left": 394,
```

```
    "top": 54
},
"faceLandmarks": {
    "pupilLeft": {
        "x": 412.7,
        "y": 78.4
    },
    "pupilRight": {
        "x": 446.8,
        "y": 74.2
    },
    "noseTip": {
        "x": 437.7,
        "y": 92.4
    },
    "mouthLeft": {
        "x": 417.8,
        "y": 114.4
    },
    "mouthRight": {
        "x": 451.3,
        "y": 109.3
    },
    "eyebrowLeftOuter": {
        "x": 397.9,
        "y": 78.5
    },
    "eyebrowLeftInner": {
        "x": 425.4,
        "y": 70.5
    },
    "eyeLeftOuter": {
        "x": 406.7,
        "y": 80.6
    },
    "eyeLeftTop": {
        "x": 412.2,
        "y": 76.2
    },
    "eyeLeftBottom": {
        "x": 413.0,
        "y": 80.1
    },
    "eyeLeftInner": {
        "x": 418.9,
        "y": 78.0
    },
    "eyebrowRightInner": {
        "x": 4.8,
        "y": 69.7
    },
    "eyebrowRightOuter": {
        "x": 5.5,
        "y": 68.5
    },
    "eyeRightInner": {
        "x": 441.5,
        "y": 75.0
    },
    "eyeRightTop": {
        "x": 446.4,
        "y": 71.7
    },
    "eyeRightBottom": {
        "x": 447.0,
        "y": 75.3
    },
    "eyeRightOuter": {
        "x": 451.7,
```

```

        },
        "y": 73.4
    },
    "noseRootLeft": {
        "x": 428.0,
        "y": 77.1
    },
    "noseRootRight": {
        "x": 435.8,
        "y": 75.6
    },
    "noseLeftAlarTop": {
        "x": 428.3,
        "y": 89.7
    },
    "noseRightAlarTop": {
        "x": 442.2,
        "y": 87.0
    },
    "noseLeftAlarOutTip": {
        "x": 424.3,
        "y": 96.4
    },
    "noseRightAlarOutTip": {
        "x": 446.6,
        "y": 92.5
    },
    "upperLipTop": {
        "x": 437.6,
        "y": 105.9
    },
    "upperLipBottom": {
        "x": 437.6,
        "y": 108.2
    },
    "underLipTop": {
        "x": 436.8,
        "y": 111.4
    },
    "underLipBottom": {
        "x": 437.3,
        "y": 114.5
    }
},
"faceAttributes": {
    "age": 71.0,
    "gender": "male",
    "smile": 0.88,
    "facialHair": {
        "mustache": 0.8,
        "beard": 0.1,
        "sideburns": 0.02
    },
    "glasses": "sunglasses",
    "headPose": {
        "roll": 2.1,
        "yaw": 3,
        "pitch": 0
    }
}
]

```

## Create a Person Group With Face API Using Python

Use the [Person Group - Create a Person Group method](#) to create a new person group with specified personGroupId, name, and user-provided userData. A person group is one of the most important parameters for the Face - Identify

API. The Identify API searches for persons' faces in a specified person group.

**Person Group - Create a Person Group Example**

```

##### Python 2.7 #####
import httplib, urllib, base64

headers = {
    # Request headers. Replace the placeholder key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

# Replace 'examplegroupid' with an ID you haven't used for creating a group before.
# The valid characters for the ID include numbers, English letters in lower case, '-' and '_'.
# The maximum length of the ID is 64.
personGroupId = 'examplegroupid'

# The userData field is optional. The size limit for it is 16KB.
body = "{ 'name':'group1', 'userData':'user-provided data attached to the person group' }"

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("POST", "/face/v1.0/persongroups/%s" % personGroupId, body, headers)
    response = conn.getresponse()

    # 'OK' indicates success. 'Conflict' means a group with this ID already exists.
    # If you get 'Conflict', change the value of personGroupId above and try again.
    # If you get 'Access Denied', verify the validity of the subscription key above and try again.
    print(response.reason)

    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e.errno, e.strerror))
#####

##### Python 3.2 #####
import http.client, urllib.request, urllib.parse, urllib.error, base64, sys

headers = {
    # Request headers. Replace the placeholder key below with your subscription key.
    'Content-Type': 'application/json',
    'Ocp-Apim-Subscription-Key': '13hc77781f7e4b19b5fcdd72a8df7156',
}

# Replace 'examplegroupid' with an ID you haven't used for creating a group before.
# The valid characters for the ID include numbers, English letters in lower case, '-' and '_'.
# The maximum length of the ID is 64.
personGroupId = 'examplegroupid'

# The userData field is optional. The size limit for it is 16KB.
body = "{ 'name':'group1', 'userData':'user-provided data attached to the person group' }"

try:
    conn = http.client.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("PUT", "/face/v1.0/persongroups/%s" % personGroupId, body, headers)
    response = conn.getresponse()

    # 'OK' indicates success. 'Conflict' means a group with this ID already exists.
    # If you get 'Conflict', change the value of personGroupId above and try again.
    # If you get 'Access Denied', verify the validity of the subscription key above and try again.
    print(response.reason)

    conn.close()
except Exception as e:
    print(e.args)
#####

```

# Face API Ruby Quick Starts

4/18/2017 • 2 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Face API with Ruby to accomplish the following tasks:

- [Detect Faces in Images](#)
- [Identify Faces in Images](#)

Learn more about obtaining free Subscription Keys [here](#).

## Detect Faces in Images With Face API Using Ruby

Use the [Face - Detect method](#) to detect faces in an image and return face attributes including:

- Face ID: Unique ID used in a number of Face API scenarios.
- Face Rectangle: The left, top, width, and height indicating the location of the face in the image.
- Landmarks: An array of 27-point face landmarks pointing to the important positions of face components.
- Facial attributes including age, gender, smile intensity, head pose, and facial hair.

### Face Detect Ruby Example Request

```
require 'net/http'

uri = URI('https://westus.api.cognitive.microsoft.com/face/v1.0/detect')
uri.query = URI.encode_www_form({
    # Request parameters
    'returnFaceId' => 'true',
    'returnFaceLandmarks' => 'false',
    'returnFaceAttributes' => '{string}'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# Request headers
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Request body
request.body = "{body}"

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

puts response.body
```

### Face - Detect Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[  
  {  
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",  
    "faceRectangle": {  
      "width": 78,  
      "height": 78,  
      "left": 394,  
      "top": 54  
    }  
  }  
]
```

```
},
"faceLandmarks": {
    "pupilLeft": {
        "x": 412.7,
        "y": 78.4
    },
    "pupilRight": {
        "x": 446.8,
        "y": 74.2
    },
    "noseTip": {
        "x": 437.7,
        "y": 92.4
    },
    "mouthLeft": {
        "x": 417.8,
        "y": 114.4
    },
    "mouthRight": {
        "x": 451.3,
        "y": 109.3
    },
    "eyebrowLeftOuter": {
        "x": 397.9,
        "y": 78.5
    },
    "eyebrowLeftInner": {
        "x": 425.4,
        "y": 70.5
    },
    "eyeLeftOuter": {
        "x": 406.7,
        "y": 80.6
    },
    "eyeLeftTop": {
        "x": 412.2,
        "y": 76.2
    },
    "eyeLeftBottom": {
        "x": 413.0,
        "y": 80.1
    },
    "eyeLeftInner": {
        "x": 418.9,
        "y": 78.0
    },
    "eyebrowRightInner": {
        "x": 4.8,
        "y": 69.7
    },
    "eyebrowRightOuter": {
        "x": 5.5,
        "y": 68.5
    },
    "eyeRightInner": {
        "x": 441.5,
        "y": 75.0
    },
    "eyeRightTop": {
        "x": 446.4,
        "y": 71.7
    },
    "eyeRightBottom": {
        "x": 447.0,
        "y": 75.3
    },
    "eyeRightOuter": {
        "x": 451.7,
        "y": 73.4
    }
},
```

```

},
"noseRootLeft": {
  "x": 428.0,
  "y": 77.1
},
"noseRootRight": {
  "x": 435.8,
  "y": 75.6
},
"noseLeftAlarTop": {
  "x": 428.3,
  "y": 89.7
},
"noseRightAlarTop": {
  "x": 442.2,
  "y": 87.0
},
"noseLeftAlarOutTip": {
  "x": 424.3,
  "y": 96.4
},
"noseRightAlarOutTip": {
  "x": 446.6,
  "y": 92.5
},
"upperLipTop": {
  "x": 437.6,
  "y": 105.9
},
"upperLipBottom": {
  "x": 437.6,
  "y": 108.2
},
"underLipTop": {
  "x": 436.8,
  "y": 111.4
},
"underLipBottom": {
  "x": 437.3,
  "y": 114.5
}
},
"faceAttributes": {
  "age": 71.0,
  "gender": "male",
  "smile": 0.88,
  "facialHair": {
    "mustache": 0.8,
    "beard": 0.1,
    "sideburns": 0.02
  },
  "glasses": "sunglasses",
  "headPose": {
    "roll": 2.1,
    "yaw": 3,
    "pitch": 0
  }
}
]

```

## Identify Faces in Images With Face API Using Ruby

Use the [Face - Identify method](#) identify people based on a detected face and people database (defined as a person group) which needs to be created in advance and can be edited over time

## Face - Identify Ruby Example Request

```
require 'net/http'

uri = URI('https://westus.api.cognitive.microsoft.com/face/v1.0/identify')
uri.query = URI.encode_www_form({})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# Request headers
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Request body
request.body = "{body}"

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

puts response.body
```

## Face - Identify Response

A successful response will be returned in JSON. Following is an example of a successful response:

```
[
  {
    "faceId": "c5c24a82-6845-4031-9d5d-978df9175426",
    "candidates": [
      {
        "personId": "25985303-c537-4467-b41d-bdb45cd95ca1",
        "confidence": 0.92
      }
    ]
  },
  {
    "faceId": "65d083d4-9447-47d1-af30-b626144bf0fb",
    "candidates": [
      {
        "personId": "2ae4935b-9659-44c3-977f-61fac20d0538",
        "confidence": 0.89
      }
    ]
  }
]
```

# Getting Started with Face API in C# Tutorial

4/12/2017 • 4 min to read • [Edit Online](#)

In this tutorial, you will learn to create and develop a simple Windows application that invokes the Face API to detect faces in an image by framing the faces.



## Preparation

To use the tutorial, you will need the following prerequisites:

- Make sure Visual Studio 2015 is installed.

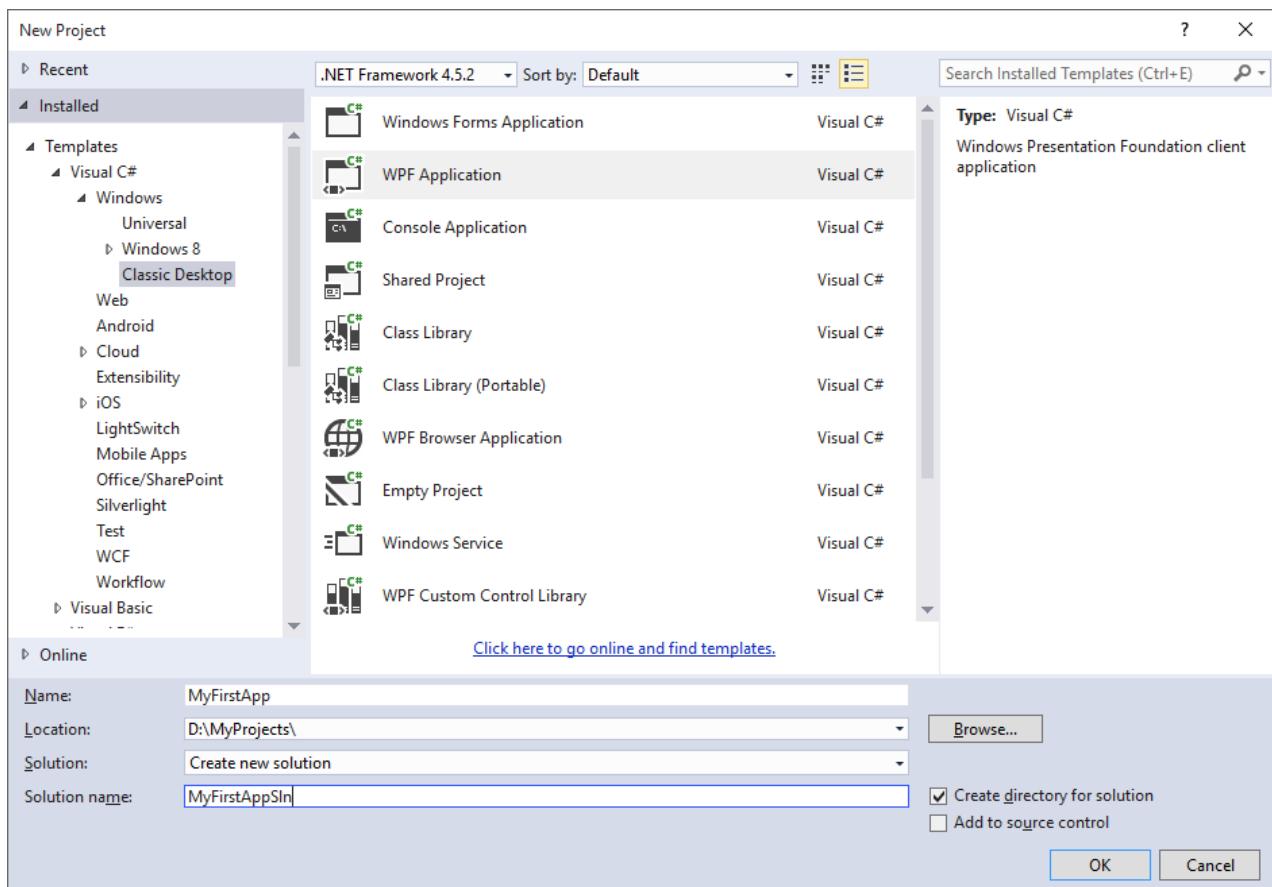
## Step 1: Subscribe for Face API and get your subscription key

Before using any Face API, you must sign up to subscribe to Face API in the Microsoft Cognitive Services (formerly Project Oxford) portal. See [subscriptions](#). Both primary and secondary key can be used in this tutorial.

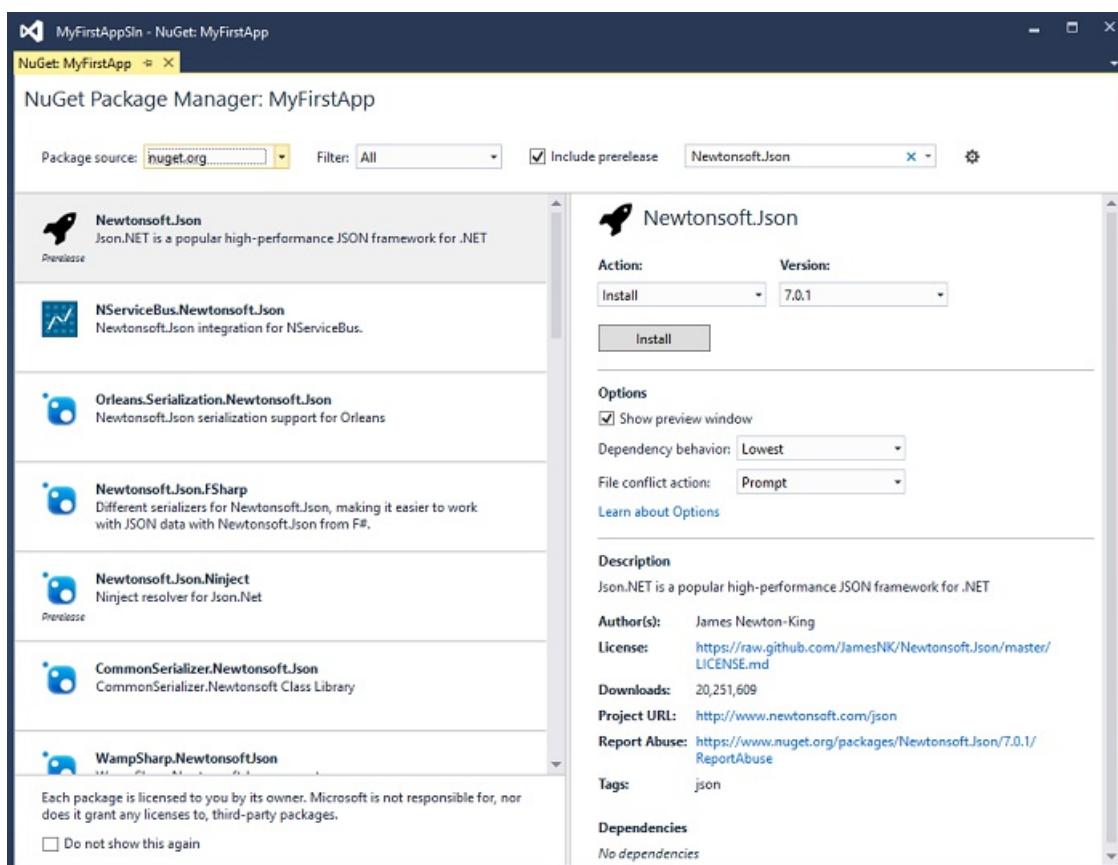
## Step 2: Create the application framework

In this step you will create a Windows application project to implement the basic UI for picking up and displaying an image. Simply follow the instructions below:

1. Open Visual Studio 2015.
2. From the File menu, click New and then Project.
3. In the New Project dialog box, click Visual C# > Windows > Classic Desktop > WPF Application.
4. Name the application *MyFirstApp*, check the 'Create directory for solution' checkbox, name the solution *MyFirstAppSln*, and then click OK.



1. Locate the Solution Explorer, right click your project (MyFirstApp in this case) and then click **Manage NuGet Packages**.
2. In NuGet Package Manager window, select nuget.org as your Package source, search for Newtonsoft.Json and install.



1. Open MainWindow.xaml, and replace the existing code with the following code to create the window UI:

```

<Window x:Class="MyFirstApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="450">
    <Grid x:Name="BackPanel">
        <Image x:Name="FacePhoto" Stretch="Uniform" Margin="0,0,0,30"/>
        <Button x:Name="BrowseButton" Margin="20,5" Height="20"
                VerticalAlignment="Bottom" Content="Browse..." 
                Click="BrowseButton_Click"/>
    </Grid>
</Window>

```

2. Open MainWindow.xaml.cs, and insert the following code inside the MainWindow class for the 'Browse' button:

```

private void BrowseButton_Click(object sender, RoutedEventArgs e)
{
    var openDlg = new Microsoft.Win32.OpenFileDialog();

    openDlg.Filter = "JPEG Image (*.jpg)|*.jpg";
    bool? result = openDlg.ShowDialog(this);

    if (!(bool)result)
    {
        return;
    }

    string filePath = openDlg.FileName;

    Uri fileUri = new Uri(filePath);
    BitmapImage bitmapSource = new BitmapImage();

    bitmapSource.BeginInit();
    bitmapSource.CacheOption = BitmapCacheOption.None;
    bitmapSource.UriSource = fileUri;
    bitmapSource.EndInit();

    FacePhoto.Source = bitmapSource;
}

```

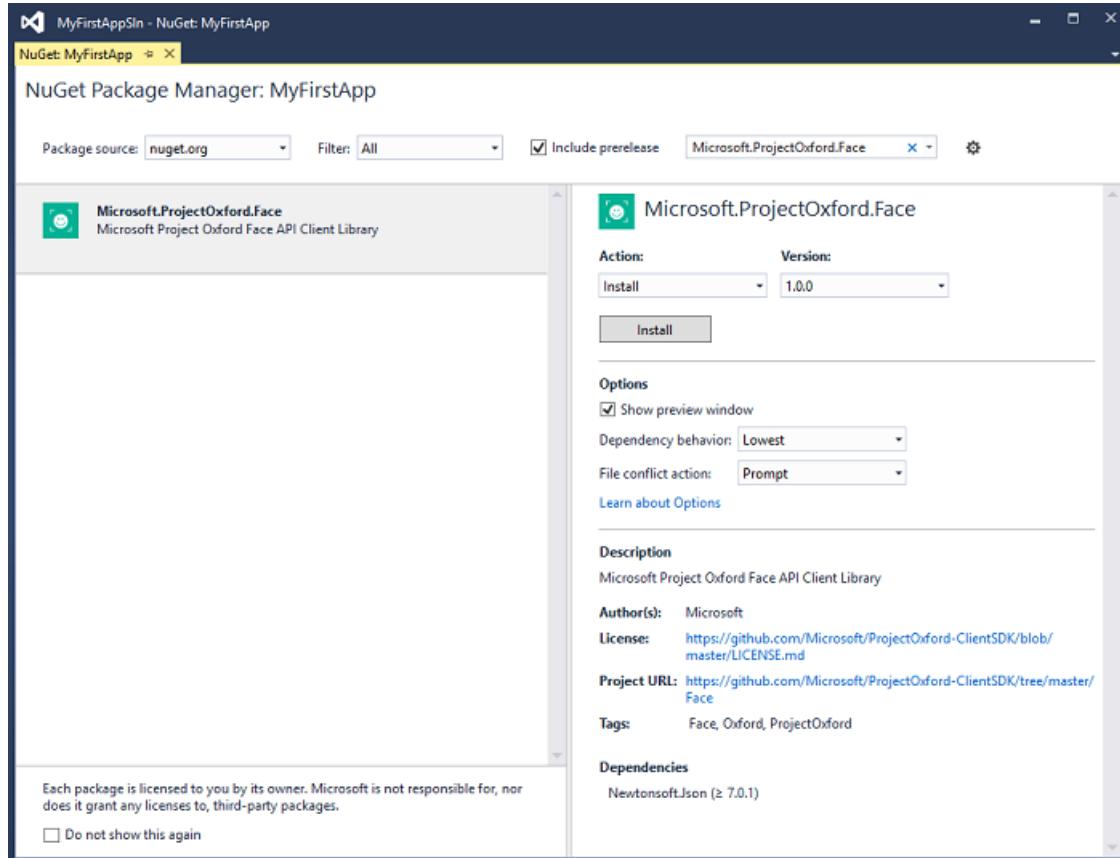
Now your app can browse for a photo and display it in the window, similar to the image below:



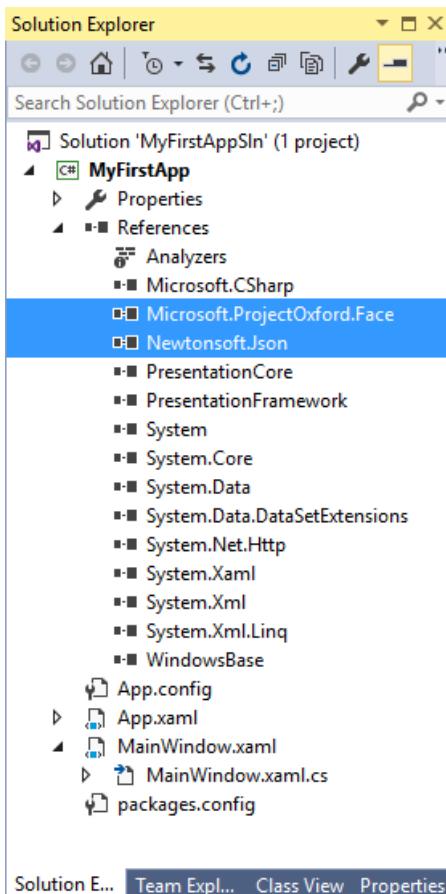
## Step 3: Configure the Face API client library

Face API is a cloud API which you can invoke through HTTPS requests. For a more convenient approach to using Face API in .NET platform applications, a client library is also provided to encapsulate the web requests. In this example, we use the client library to simplify our work. Follow the instructions below to configure the client library:

1. Locate the Solution Explorer, right click your project (MyFirstApp in this case) and then click Manage NuGet Packages.
2. In the NuGet Package Manager window, select nuget.org as your Package source, search for Microsoft.ProjectOxford.Face and install.



1. Check your project references, Microsoft.ProjectOxford.Face will be automatically added after the installation succeeds.



1. Open MainWindow.xaml.cs in your MyFirstApp project, add this using directives to the beginning of the file:  
using System.IO; using Microsoft.ProjectOxford.Face; using Microsoft.ProjectOxford.Face.Contract;
2. Insert the following code in the MainWindow class: private readonly IFaceServiceClient faceServiceClient = new FaceServiceClient("key"); Replace the word *key* with the subscription key you obtained in step 1.
3. Now you are ready to call the Face API from your application.

## Step 4: Upload images to detect faces

The most straightforward way to detect faces is by calling the [Face - Detect](#) API by uploading the image file directly. When using the client library, this can be done by using the asynchronous method DetectAsync of FaceServiceClient. Each returned face contains a rectangle to indicate its location, combined with a series of optional face attributes. In this example, we only need to retrieve the face location. Here we need to insert an asynchronous function into the MainWindow class for face detection:

```
private async Task<FaceRectangle[]> UploadAndDetectFaces(string imagePath)
{
    try
    {
        using (Stream imageFileStream = File.OpenRead(imagePath))
        {
            var faces = await faceServiceClient.DetectAsync(imageFileStream);
            var faceRects = faces.Select(face => face.FaceRectangle);
            return faceRects.ToArray();
        }
    }
    catch (Exception)
    {
        return new FaceRectangle[0];
    }
}
```

## Step 5: Mark faces in the image

In this last step, we combine all the above steps and mark the detected faces in the image. First, open MainWindow.xaml.cs and add the 'async' modifier to the BrowseButton\_Click method:

```
private async void BrowseButton_Click(object sender, RoutedEventArgs e)
```

Now insert the following code at the end of the BrowseButton\_Click event handler:

```
Title = "Detecting...";  
FaceRectangle[] faceRects = await UploadAndDetectFaces(filePath);  
Title = String.Format("Detection Finished. {0} face(s) detected", faceRects.Length);  
  
if (faceRects.Length > 0)  
{  
    DrawingVisual visual = new DrawingVisual();  
    DrawingContext drawingContext = visual.RenderOpen();  
    drawingContext.DrawImage(bitmapSource,  
        new Rect(0, 0, bitmapSource.Width, bitmapSource.Height));  
    double dpi = bitmapSource.DpiX;  
    double resizeFactor = 96 / dpi;  
  
    foreach (var faceRect in faceRects)  
    {  
        drawingContext.DrawRectangle(  
            Brushes.Transparent,  
            new Pen(Brushes.Red, 2),  
            new Rect(  
                faceRect.Left * resizeFactor,  
                faceRect.Top * resizeFactor,  
                faceRect.Width * resizeFactor,  
                faceRect.Height * resizeFactor  
            )  
        );  
    }  
  
    drawingContext.Close();  
    RenderTargetBitmap faceWithRectBitmap = new RenderTargetBitmap(  
        (int)(bitmapSource.PixelWidth * resizeFactor),  
        (int)(bitmapSource.PixelHeight * resizeFactor),  
        96,  
        96,  
        PixelFormats.Pbgra32);  
  
    faceWithRectBitmap.Render(visual);  
    FacePhoto.Source = faceWithRectBitmap;  
}
```

Run this application and browse for an image containing a face. Please wait for a few seconds to allow the cloud API to respond. After that, you will get a result similar to the image below:



## Summary

In this tutorial, you have learned the basic process for using the Face API and created an application to display face marks in images. For more information on API details, please refer to the How-To and [API Reference](#).

## Related Topics

- [Getting Started with Face API in Java for Android](#)
- [Getting Started with Face API in Python](#)

# Getting Started with Face API in Java for Android Tutorial

4/12/2017 • 6 min to read • [Edit Online](#)

In this tutorial, you will learn to create and develop a simple Android application that invokes the Face API to detect human faces in an image. The application shows the result by framing the faces that it detects.



## Preparation

To use the tutorial, you will need the following prerequisites:

- Android Studio and SDK installed
- Android device (optional for testing)

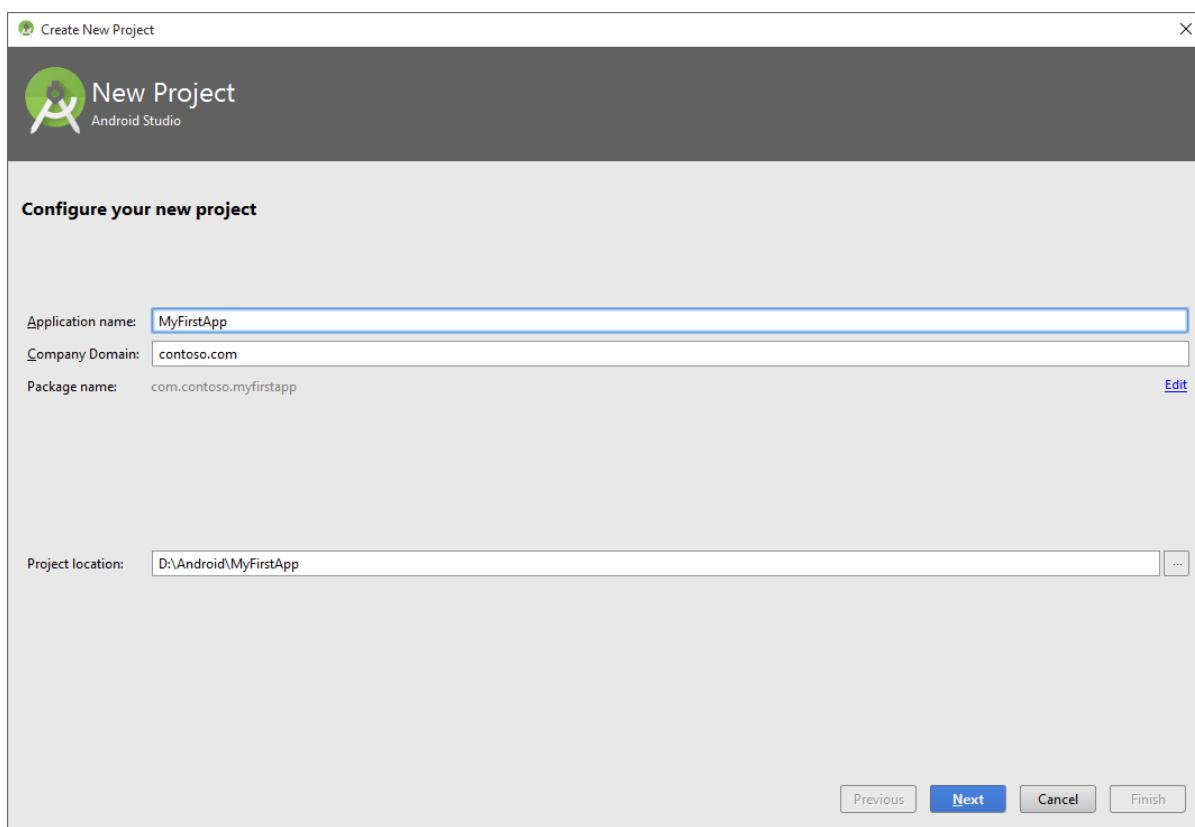
## Step 1: Subscribe for Face API and get your subscription key

Before using any Face API, you must sign up to subscribe to Face API in the Microsoft Cognitive Services (formerly Project Oxford) portal. See [subscriptions](#). Both primary and secondary key can be used in this tutorial.

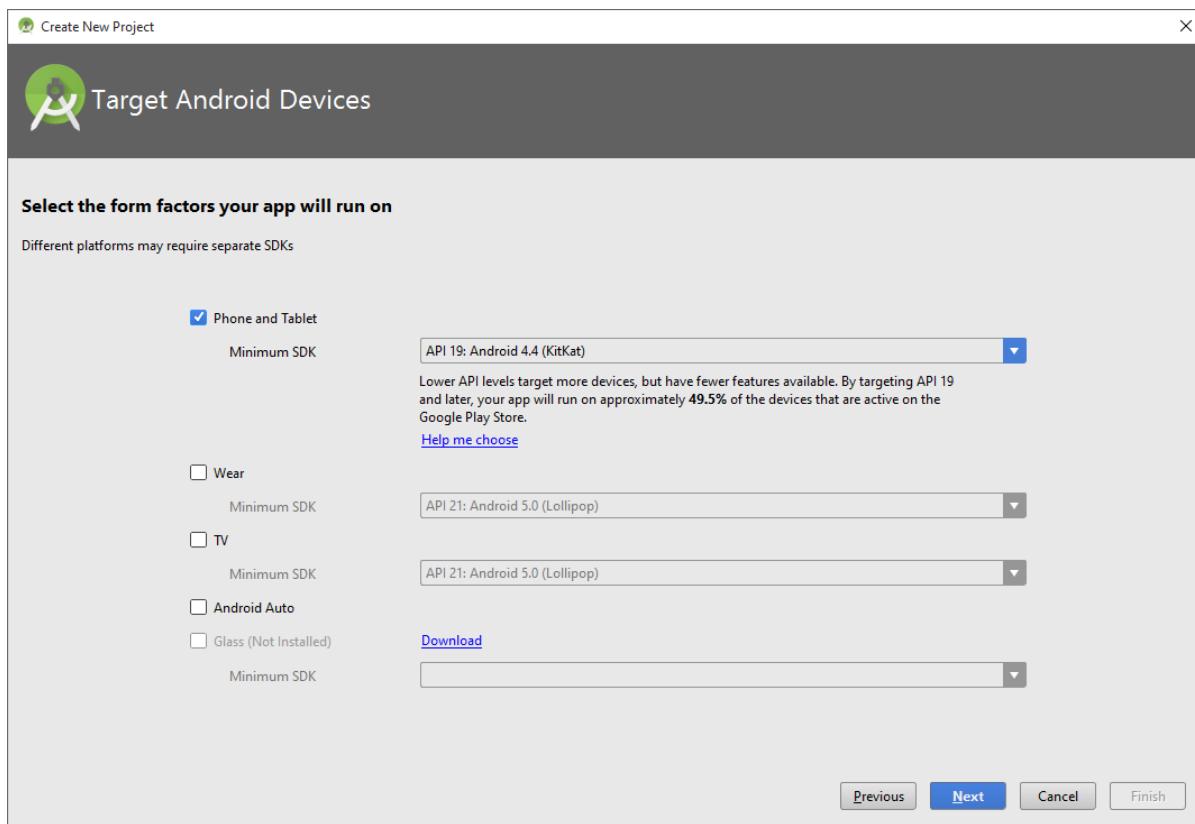
## Step 2: Create the application framework

In this step you will create an Android application project to implement the basic UI for picking up and displaying an image. Simply follow the instructions below:

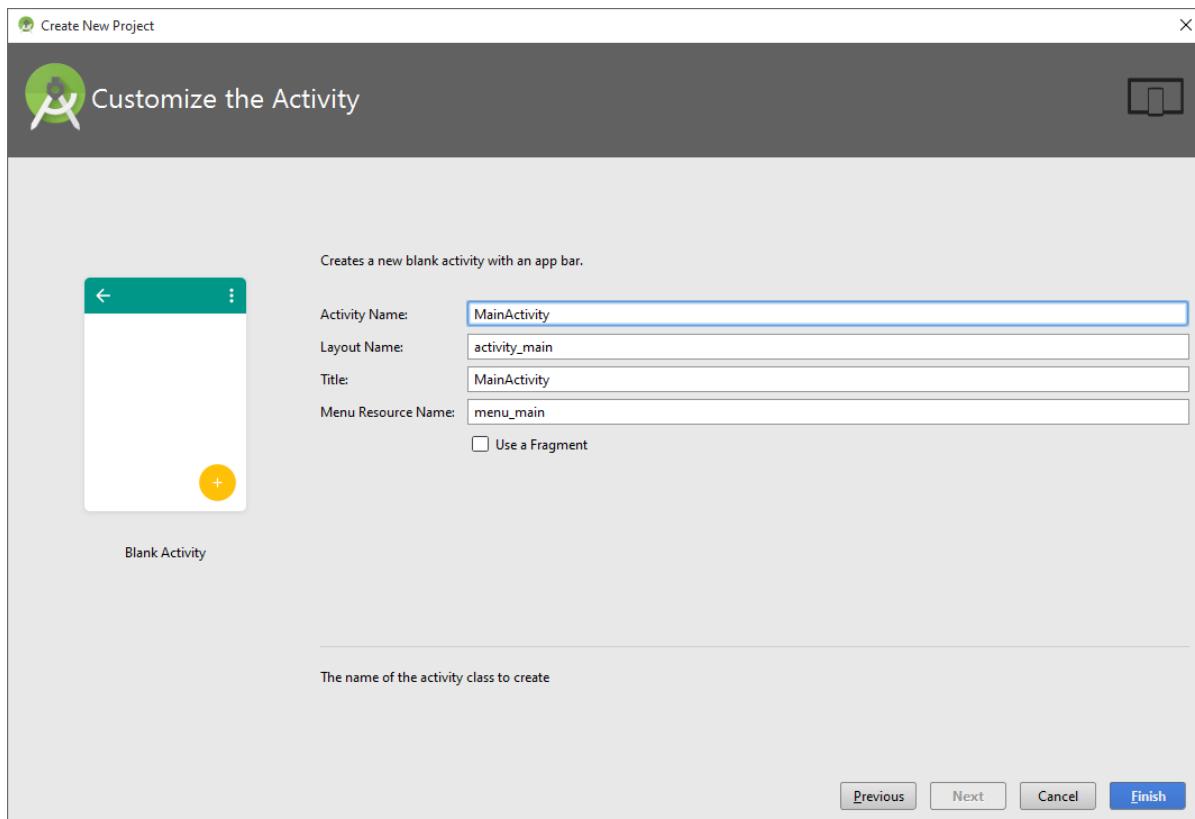
1. Open Android Studio.
2. From the File menu, click New Project...
3. Name the application MyFirstApp, and then click Next.



4. Choose target platform as required, and then click Next.



5. Select "Basic Activity" and then click Next.
6. Name the activity as follows, and then click Finish.



7. Open activity\_main.xml, you should see the Layout Editor of this activity.
8. View Text source file and then edit the activity layout as follows:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:id="@+id/imageView1"
        android:layout_above="@+id/button1" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Browse"
        android:id="@+id/button1"
        android:layout_alignParentBottom="true" />
</RelativeLayout>
```

9. Open MainActivity.java and insert the following import directives at the beginning of the file:

```
import java.io.*;
import android.app.*;
import android.content.*;
import android.net.*;
import android.os.*;
import android.view.*;
import android.graphics.*;
import android.widget.*;
import android.provider.*;
```

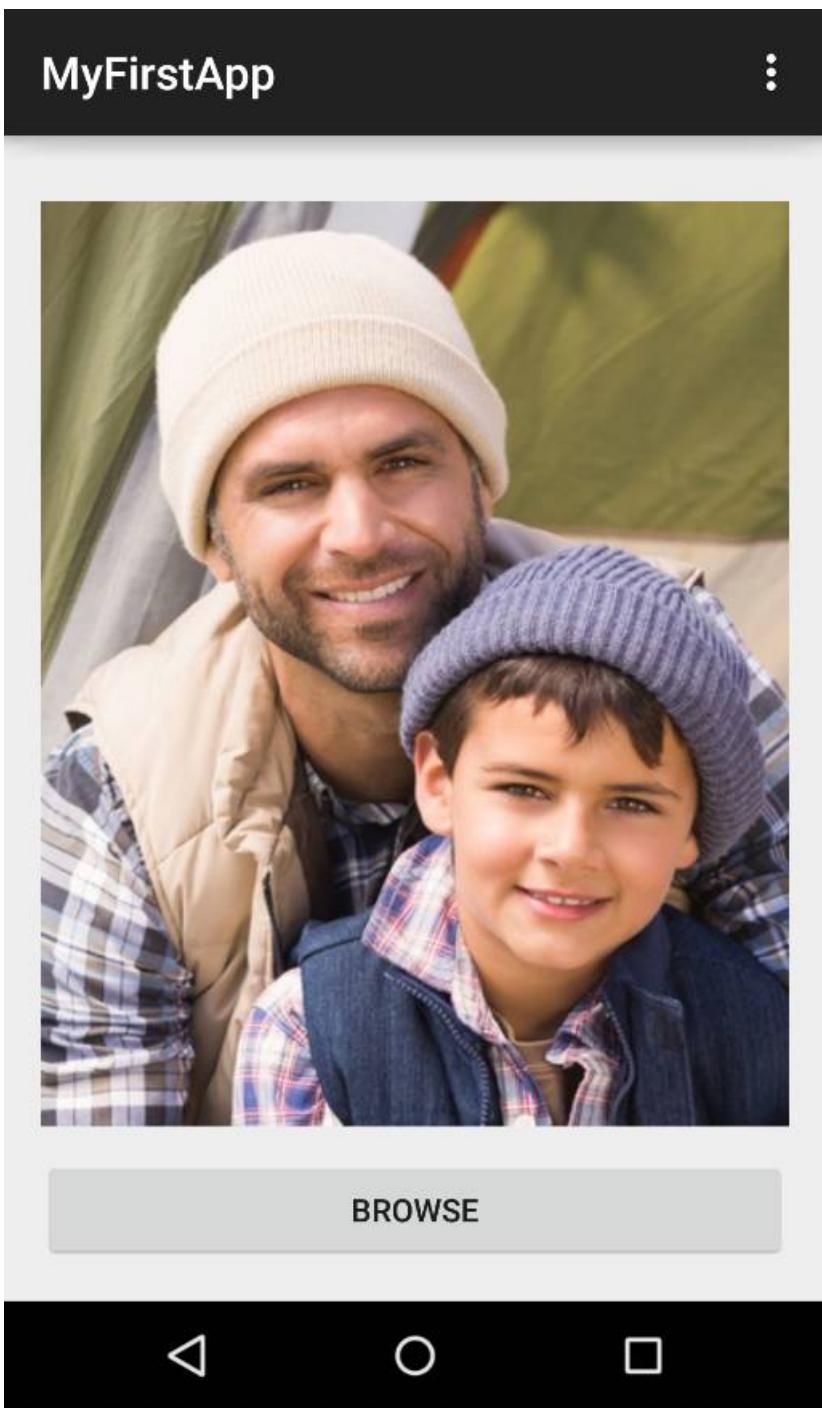
Secondly, Modify the onCreate method of the MainActivity class for the 'Browse' button logic:

```
private final int PICK_IMAGE = 1;
private ProgressDialog detectionProgressDialog;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button button1 = (Button)findViewById(R.id.button1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent gallIntent = new Intent(Intent.ACTION_GET_CONTENT);
            gallIntent.setType("image/*");
            startActivityForResult(Intent.createChooser(gallIntent, "Select Picture"), PICK_IMAGE);
        }
    });
}

detectionProgressDialog = new ProgressDialog(this);
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == PICK_IMAGE && resultCode == RESULT_OK && data != null && data.getData() != null)
    {
        Uri uri = data.getData();
        try {
            Bitmap bitmap = MediaStore.Images.Media.getBitmap(getApplicationContext(), uri);
            ImageView imageView = (ImageView) findViewById(R.id.imageView1);
            imageView.setImageBitmap(bitmap);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Now your app can browse for a photo from the gallery and display it in the window similar to the image below:



## Step 3: Configure the Face API client library

The Face API is a cloud API which you can invoke using HTTPS requests. For a more convenient way of using the Face API in .NET platform applications, a client library is also provided to encapsulate the web requests. In this example, we use the client library to simplify our work.

Follow the instructions below to configure the client library:

1. Locate the top-level build.gradle file of your project from the Project panel shown in the example. Note that there are several other build.gradle files in your project tree, and you need to open the top-level build.gradle file at first.
2. Add mavenCentral() to your projects' repositories. You can also use jcenter(), which is the default repository of Android Studio, since jcenter() is a superset of mavenCentral().

```
allprojects {
    repositories {
        ...
        mavenCentral()
    }
}
```

3. Open the build.gradle file in your 'app' project.
4. Add a dependency for our client library stored in the Maven Central Repository:

```
dependencies {
    ...
    compile 'com.microsoft.projectoxford:face:1.0.0'
}
```

5. Open MainActivity.java in your 'app' project and insert the following import directives:

```
import com.microsoft.projectoxford.face.*;
import com.microsoft.projectoxford.face.contract.*;
```

And then insert the following code in the MainActivity class:

```
private FaceServiceClient faceServiceClient =
    new FaceServiceRestClient("your subscription key");
```

Replace the string above with the subscription key you obtained in step 1.

6. Open the file called AndroidManifest.xml in your 'app' project (in the app/src/main directory). Insert the following element into the manifest element:

```
<uses-permission android:name="android.permission.INTERNET" />
```

7. Now you are ready to call the Face API from your application.

## Step 4: Upload images to detect faces

The most straightforward way to detect faces is by calling the [Face – Detect](#) API by uploading the image file directly. When using the client library, this can be done by using an asynchronous method DetectAsync of FaceServiceClient. Each returned face contains a rectangle to indicate its location, combined with a series of optional face attributes. In this example, we only need to retrieve the face location. Here we need to insert a method into the MainActivity class for face detection:

```

// Detect faces by uploading face images
// Frame faces after detection

private void detectAndFrame(final Bitmap imageBitmap)
{
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    imageBitmap.compress(Bitmap.CompressFormat.JPEG, 100, outputStream);
    ByteArrayInputStream inputStream =
        new ByteArrayInputStream(outputStream.toByteArray());
    AsyncTask<InputStream, String, Face[]> detectTask =
        new AsyncTask<InputStream, String, Face[]>() {
            @Override
            protected Face[] doInBackground(InputStream... params) {
                try {
                    publishProgress("Detecting...");
                    Face[] result = faceServiceClient.detect(
                        params[0],
                        true,           // returnFaceId
                        false,          // returnFaceLandmarks
                        null           // returnFaceAttributes: a string like "age, gender"
                    );
                    if (result == null)
                    {
                        publishProgress("Detection Finished. Nothing detected");
                        return null;
                    }
                    publishProgress(
                        String.format("Detection Finished. %d face(s) detected",
                            result.length));
                    return result;
                } catch (Exception e) {
                    publishProgress("Detection failed");
                    return null;
                }
            }
            @Override
            protected void onPreExecute() {
                //TODO: show progress dialog
            }
            @Override
            protected void onProgressUpdate(String... progress) {
                //TODO: update progress
            }
            @Override
            protected void onPostExecute(Face[] result) {
                //TODO: update face frames
            }
        };
    detectTask.execute(inputStream);
}

```

## Step 5: Mark faces in the image

In this last step, we combine all the above steps together and mark the detected faces with frames in the image. First, open `MainActivity.java` and insert a helper method in `MainActivity.java` to draw rectangles:

```

private static Bitmap drawFaceRectanglesOnBitmap(Bitmap originalBitmap, Face[] faces) {
    Bitmap bitmap = originalBitmap.copy(Bitmap.Config.ARGB_8888, true);
    Canvas canvas = new Canvas(bitmap);
    Paint paint = new Paint();
    paint.setAntiAlias(true);
    paint.setStyle(Paint.Style.STROKE);
    paint.setColor(Color.RED);
    int strokeWidth = 2;
    paint.setStrokeWidth(strokeWidth);
    if (faces != null) {
        for (Face face : faces) {
            FaceRectangle faceRectangle = face.faceRectangle;
            canvas.drawRect(
                faceRectangle.left,
                faceRectangle.top,
                faceRectangle.left + faceRectangle.width,
                faceRectangle.top + faceRectangle.height,
                paint);
        }
    }
    return bitmap;
}

```

Now finish the TODO parts in the detectAndFrame method in order to frame faces and report status.

```

@Override
protected void onPreExecute() {

    detectionProgressDialog.show();
}

@Override
protected void onProgressUpdate(String... progress) {

    detectionProgressDialog.setMessage(progress[0]);
}

@Override
protected void onPostExecute(Face[] result) {

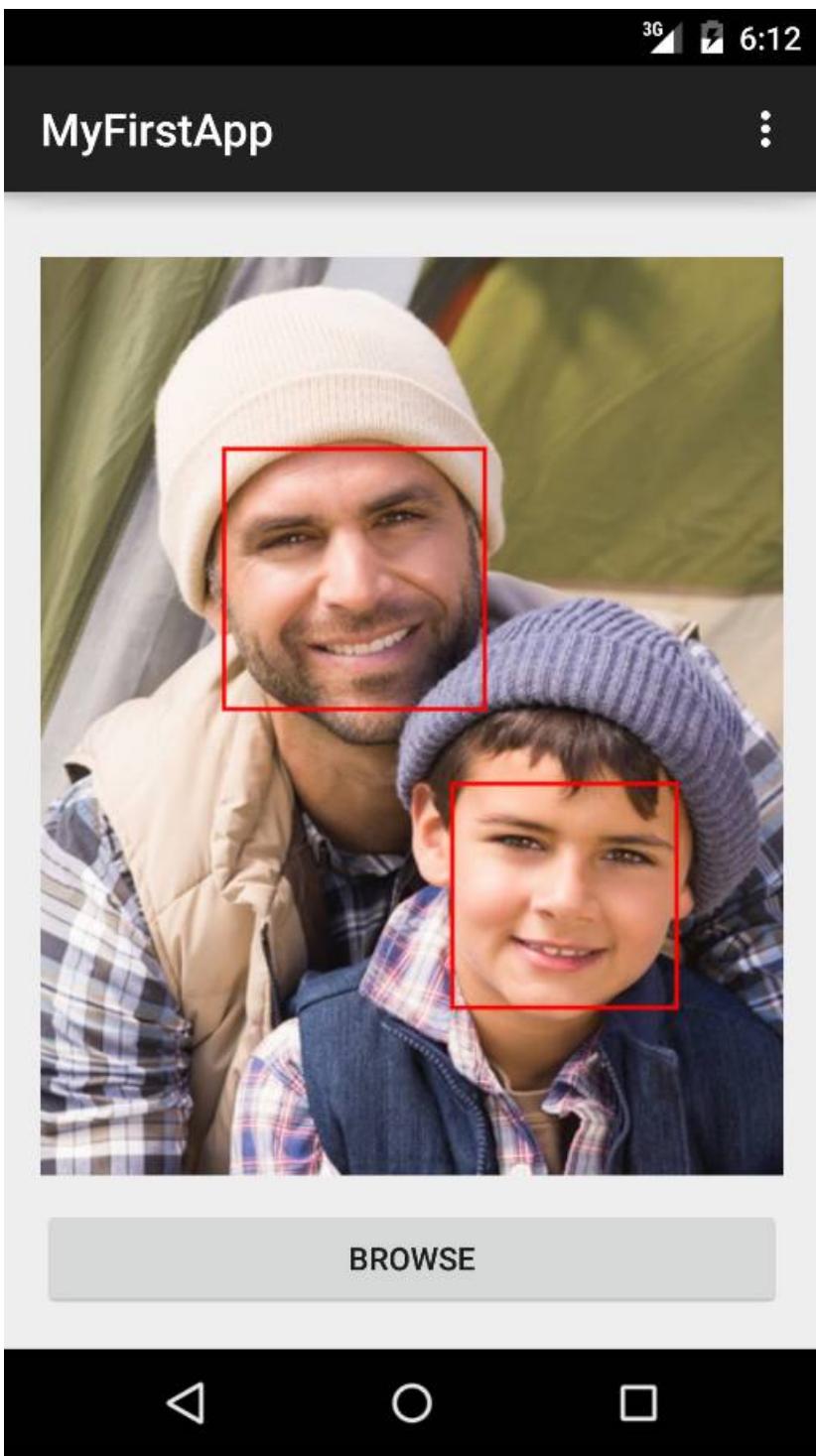
    detectionProgressDialog.dismiss();
    if (result == null) return;
    ImageView imageView = (ImageView) findViewById(R.id.imageView1);
    imageView.setImageBitmap(drawFaceRectanglesOnBitmap(imageBitmap, result));
    imageBitmap.recycle();
}

```

Finally, add a call to the detectAndFrame method from the onActivityResult method, as shown below. (Note that the asterisks are only intended to highlight the new addition. You must remove them before attempting to build the code.)

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
    if (requestCode == PICK_IMAGE && resultCode == RESULT_OK && data != null && data.getData() != null) {  
        Uri uri = data.getData();  
        try {  
            Bitmap bitmap = MediaStore.Images.Media.getBitmap(getApplicationContext(), uri);  
            ImageView imageView = (ImageView) findViewById(R.id.imageView1);  
            imageView.setImageBitmap(bitmap);  
  
            **detectAndFrame(bitmap);**  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Run this application and browse for an image containing a face. Please wait for a few seconds to allow the cloud API to respond. After that, you will get a result similar to the image below:



## Summary

In this tutorial, you learned the basic process for using the Face API and created an application to display face marks in images. For more information on the Face API, refer to the How-To and [API Reference](#).

## Related Tutorials

- [Getting Started with Face API in CSharp Tutorial](#)
- [Getting Started with Face API in Python Tutorial](#)

# Getting Started with Face API in Python Tutorial

4/12/2017 • 1 min to read • [Edit Online](#)

In this tutorial, you will learn to invoke the Face API via the Python SDK to detect human faces in an image.

## Prerequisites

To use the tutorial, you will need to do the following:

- Install either Python 2.7+ or Python 3.5+.
- Install pip.
- Install the Python SDK for the Face API as follows:

```
pip install cognitive_face
```

- Obtain a [subscription key](#) for Microsoft Cognitive Services (formerly Project Oxford). You can use either your primary or your secondary key in this tutorial. (Note that to use any Face API, you must have a valid subscription key.)

## Detect a Face in an Image

```
import cognitive_face as CF

KEY = 'subscription key' # Replace with a valid subscription key (keeping the quotes in place).
CF.Key.set(KEY)

# You can use this example JPG or replace the URL below with your own URL to a JPEG image.
img_url = 'https://raw.githubusercontent.com/Microsoft/Cognitive-Face-Windows/master/Data/detection1.jpg'
result = CF.face.detect(img_url)
print result
```

Below is an example result. It's a `list` of detected faces. Each item in the list is a `dict` instance where `faceId` is a unique ID for the detected face and `faceRectangle` describes the position of the detected face. A face ID expires in 24 hours.

```
[{u'faceId': u'68a0f8cf-9dba-4a25-afb3-f9cdf57cca51', u'faceRectangle': {u'width': 89, u'top': 66, u'height': 89, u'left': 446}}]
```

## Further Exploration

To help you further explore the Face API, this tutorial provides a GUI sample. To run it, first install [wxPython](#) then run the commands below.

```
git clone https://github.com/Microsoft/Cognitive-Face-Python.git
cd Cognitive-Face-Python
python sample
```

## Summary

In this tutorial, you have learned the basic process for using the Face API via invoking the Python SDK. For more information on API details, please refer to the How-To and [API Reference](#).

## Related Topics

- [Getting Started with Face API in CSharp](#)
- [Getting Started with Face API in Java for Android](#)

# API Reference

4/12/2017 • 1 min to read • [Edit Online](#)

The Microsoft Face API is a cloud-based API that provides the most advanced algorithms for face detection and recognition. The reference document can be found in [Microsoft Cognitive Services \(formerly Project Oxford\) Dev Portal](#).

Face APIs cover the following categories:

- [Person Management APIs](#): Used to manage person faces for [Identification](#).
- [Person Group Management APIs](#): Used to manage a person dataset for [Identification](#).
- [Face List Management APIs](#): Used to manage a face list for [Find Similar](#).
- [Face Algorithm APIs](#): Covers core functions such as [Detection](#), [Verification](#), and [Identification](#).

# Glossary

4/12/2017 • 9 min to read • [Edit Online](#)

## A

### **Attributes**

Attributes are optional in the [detection](#) results, such as [age](#), [gender](#), [head pose](#), [facial hair](#), [smiling](#). They can be obtained from the [detection](#) API by specifying the query parameters: `returnFaceAttributes`. Attributes give extra information regarding selected [faces](#); in addition to the [face ID](#) and the [rectangle](#).

For more details, please refer to the guide [Face - Detect](#).

### **Age (Attribute)**

Age is one of the [attributes](#) that describes the age of a particular face. The age attribute is optional in the [detection](#) results, and can be controlled with a [detection](#) request by specifying the `returnFaceAttributes` parameter.

For more details, please refer to the guide [Face - Detect](#).

## B

## C

### **Candidate**

Candidates are essentially [identification](#) results (e.g. identified persons and level of confidence in detections). A candidate is represented by the [personID](#) and [confidence](#), indicating that the person is identified with a high level of confidence.

For more details, please refer to the guide [Face - Identify](#).

### **Confidence**

Confidence is a measurement revealing the similarity between [faces](#) or [people](#) in numerical values –which is used in [identification](#), and [verification](#) to indicate the similarities of searched, identified and verified results.

For more details, please refer to the following guides: [Face - Find Similar](#), [Face - Identify](#), [Face - Verify](#)

## D

### **Detection/Face Detection**

Face detection is the action of locating faces in images. Users can upload an image or specify an image URL in the request. The detected faces are returned with [face IDs](#) indicating a unique identity in Face API. The rectangles indicate the face locations in the image in pixels, as well as the optional [attributes](#) for each face such as [age](#), [gender](#), [head pose](#), [facial hair](#) and [smiling](#).

For more details, please refer to the guide [Face - Detect](#).

## E

## F

### **Face**

Face is a unified term for the results derived from Face API related with detected faces. Ultimately, face is represented by a unified identity ([Face ID](#)), a specified region in images ([Face Rectangle](#)), and extra face related

attributes, such as [age](#), [gender](#), [landmarks](#) and [head pose](#). Additionally, faces can be returned from [detection](#).

For more details, please refer to the guide [Face - Detect](#).

#### Face API

Face API is a cloud-based API that provides the most advanced algorithms for face detection and recognition. The main functionality of Face API can be divided into two categories: face [detection](#) with [attributes](#), and face [recognition](#).

For more details, please refer to the following guides: [Face API Overview](#), [Face - Detect](#), [Face - Find Similar Faces](#), [Face - Group](#), [Face - Identify](#), [Face - Verify](#)

#### Face Attributes/Facial Attributes

Please see [Attributes](#).

#### Face ID

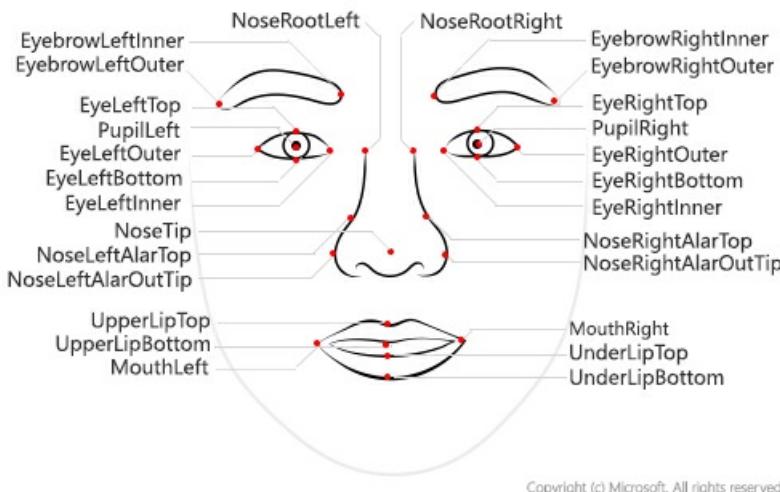
Face ID is derived from the [detection](#) results, in which a string represents a [face](#) in [Face API](#).

For more details, please refer to the guide [Face - Detect](#).

#### Face Landmarks/Facial Landmarks

Landmarks are optional in the [detection](#) results; which are semantic facial points, such as the eyes, nose and mouth (illustrated in following figure). Landmarks can be controlled with a [detection](#) request by the Boolean number `returnFaceLandmarks`. If `returnFaceLandmarks` is set as true, the returned faces will have landmark attributes.

For more details, please refer to the guide [Face - Detect](#).



#### Face Rectangle

Face rectangle is derived from the [detection](#) results, which is an upright rectangle (left, top, width, height) in images in pixels. The top-left corner of a [face](#) (left, top), besides the width and height, indicates face sizes in x and y axes respectively.

For more details, please refer to the guide [Face - Detect](#).

#### Facial Hair (Attribute)

Facial hair is one of the [attributes](#) used to describe the facial hair length of the available faces. The facial hair attribute is optional in the [detection](#) results, and can be controlled with a [detection](#) request by `returnFaceAttributes`. If `returnFaceAttributes` contains 'facialHair', the returned faces will have facial hair attributes.

For more details, please refer to the guide [Face - Detect](#).

#### Find Similar Faces

This API is used to search/query similar faces based on a collection of faces. Query faces and face collections are represented as [face IDs](#) in the request. Returned results are searched similar faces, represented by [face IDs](#).

For more details, please refer to the guide [Face - Find Similar Faces](#).

## G

### Gender (Attribute)

Gender is one of the [attributes](#) used to describe the genders of the available faces. The gender attribute is optional in the [detection](#) results, and can be controlled with a [detection](#) request by returnFaceAttributes. If returnFaceAttributes contains 'gender', the returned faces will have gender attributes.

For more details, please refer to the guide [Face - Detect](#).

### Grouping

Face grouping is the grouping of a collection of faces according to facial similarities. Face collections are indicated by the face ID collections in the request. As a result of grouping, similar faces are grouped together as [groups](#), and faces that are not similar to any other face are merged together as a messy group. There is at the most, one [messy group](#) in the grouping result.

For more details, please refer to the guide [Face - Group](#).

### Groups

Groups are derived from the [grouping](#) results. Each group contains a collection of similar faces, where faces are indicated by [face IDs](#).

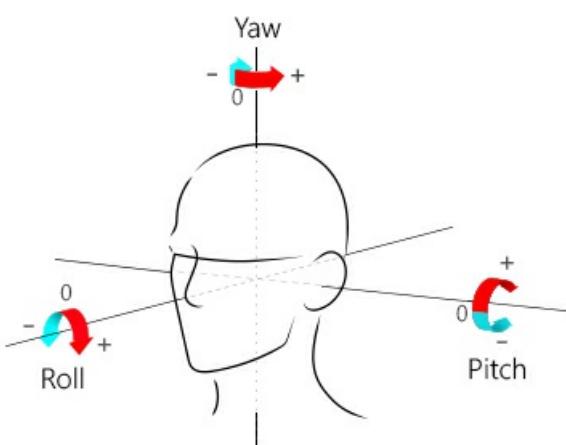
For more details, please refer to the guide [Face - Group](#).

## H

### Head Pose (Attribute)

Head pose is one of the [attributes](#) that represents face orientation in 3D space according to roll, pitch and yaw angles, as shown in following figure. The value ranges of roll and yaw are [-180, 180] and [-90, 90] in degrees. In the current version, the pitch value returned from detection is always 0. The head pose attribute is optional in the [detection](#) results, and can be controlled with a [detection](#) request by the returnFaceAttributes parameter. If returnFaceAttributes parameter contains 'headPose', the returned faces will have head pose attributes.

For more details, please refer to the guide [Face - Detect](#).



Copyright (c) Microsoft. All rights reserved

## I

### Identification

Identification is to identify one or more faces from a person group. A [person group](#) is a collection of [persons](#). Faces and the person group are represented by [face IDs](#) and [person group IDs](#) respectively in the request. Identified results are [candidates](#), represented by [persons](#) with confidence. Multiple faces in the input are considered

separately, and each face will have its own identified result.

**Please Note:** the person group should be trained successfully before identification. If the person group is not trained, or the training [status](#) is not shown as 'succeeded' (i.e. 'running', 'failed', or 'timeout'), the request response is 400.

For more details, please refer to the following guides:

[Face - Identify](#)

[Person - Create a Person](#)

[Person Group - Create a Person Group](#)

[Person Group - Train Person Group](#)

#### **IsIdentical**

`IsIdentical` is a Boolean field of [verification](#) results indicating whether two faces belong to the same person.

For more details, please refer to the guide [Face - Verify](#).

J

K

L

#### **Landmarks**

Please see [face landmarks](#).

M

#### **Messy group**

Messy group is derived from the [grouping](#) results; which contains faces not similar to any other face. Each face in a messy group is indicated by the [face ID](#).

For more details, please refer to the guide [Face - Group](#).

N

#### **Name (Person)**

Name is a user friendly descriptive string for [Person](#). Unlike the [Person ID](#), the name of people can be duplicated within a group.

For more details, please refer to the following guides:

[Person - Create a Person](#)

[Person - Get a Person](#)

#### **Name (Person Group)**

Name is also a user friendly descriptive string for [Person Group](#). Unlike the [Person Group ID](#), the name of person groups can be duplicated within a subscription.

For more details, please refer to the following guides:

[Person Group - Create a Person Group](#)

[Person Group - Get a Person Group](#)

O

P

## **Person**

Person is a data structure managed in Face API. Person comes with a [Person ID](#), as well as other attributes such as [Name](#), a collection of [Face IDs](#), and [User Data](#).

For more details, please refer to the following guides:

[Person - Create a Person](#)

[Person - Get a Person](#)

## **Person ID**

Person ID is generated when a [person](#) is created successfully. A string is created to represent this person in [Face API](#).

For more details, please refer to the following guides:

[Person - Create a Person](#)

[Person - Get a Person](#)

## **Person Group**

Person group is a collection of [Persons](#) and is the unit of [Identification](#). A person group comes with a [Person Group ID](#), as well as other attributes such as [Name](#) and [User Data](#).

For more details, please refer to the following guides:

[Person Group - Create a Person Group](#)

[Person Group - Get a Person Group](#)

[Person - List Persons in a PersonGroup](#)

## **Person Group ID**

Person group ID is a user provided string used as an identifier of a [person group](#). The group ID must be unique within the subscription.

For more details, please refer to the following guides:

[PersonGroup - Create a PersonGroup](#)

[PersonGroup - Get a PersonGroup](#)

## **Pose (Attribute)**

Please see [Head Pose](#).

# **Q**

# **R**

## **Recognition**

Recognition is a popular application area for face technologies, such as [Find Similar Faces](#), [Grouping](#), [Identify](#),[verifying two faces same or not](#).

For more details, please refer to the following guides:

[Face - Find Similar Faces](#)

[Face - Group](#)

[Face - Identify](#)

[Face - Verify](#)

## **Rectangle (Face)**

Please see [face rectangle](#).

# **S**

## **Smile (Attribute)**

Smile is one of the [attributes](#) used to describe the smile expression of the available faces. The smile attribute is

optional in the [detection](#) results, and can be controlled with a [detection](#) request by returnFaceAttributes. If returnfaceAttributes contains 'smile', the returned faces will have smile attributes.

For more details, please refer to the guide [Face - Detect](#).

#### **Similar Face Searching**

Please see [Find Similar Faces](#).

#### **Status (Train)**

Status is a string used to describe the procedure for [Training Person Groups](#), including 'notstarted', 'running', 'succeeded', 'failed'.

For more details, please refer to the guide [Person Group - Train Person Group](#).

#### **Subscription key**

Subscription key is a string that you need to specify as a query string parameter in order to invoke any Face API. The subscription key can be found in My Subscriptions page after you sign in to Microsoft Cognitive Services portal. There will be two keys associated with each subscription: one primary key and one secondary key. Both can be used to invoke API identically. You need to keep the subscription keys secure, and you can regenerate subscription keys at any time from My Subscriptions page as well.

## T

#### **Train (Person Group)**

This API is used to train a particular model for a specified [person group](#) to facilitate identification. If the training is not operated, or the [Training Status](#) is not shown as succeeded, the identification for this person group will result in failure.

For more details, please refer to the following guides: [Person Group - Train Person Group](#), [Face - Identify](#)

## U

#### **UserData/User Data**

User data is extra information associated with [person](#) and [person group](#). User data is set by users to make data easier to use, understand and remember.

For more details, please refer to the following guides:

[Person - Create a Person](#)

[Person Group - Create a Person Group](#)

[Person - Update a Person](#)

[Person Group - Update a Person Group](#)

## V

#### **Verification**

This API is used to verify whether two faces are the same or not. Both faces are represented as face IDs in the request. Verified results contain a Boolean field ([isIdentical](#)) indicating same if true and a number field ([confidence](#)) indicating the level of confidence.

For more details, please refer to the guide [Face - Verify](#).

## W

## X

## Y

Z

# Face API Frequently Asked Questions

4/12/2017 • 1 min to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, try asking the Face API community on [StackOverflow](#) or contact Help and Support on [UserVoice](#).

**Question:** What factors can reduce Face API's accuracy for Recognition, Verification, or Find Similar?

**Answer:** Generally it is the same cases where humans have difficulty identifying someone including;

- Obstructions blocking one or both eyes
- Harsh lighting, e.g. severe backlighting
- Changes to hair style or facial hair
- Changes due to age
- Extreme facial expressions (e.g. screaming)

Face API is often successful in challenging cases like these, but accuracy can be reduced. To make recognition more robust and address these challenges, train your Persons with photos that include a diversity of angles and lighting.

**Question:** I am passing the binary image data in but I get an "Invalid face image" error.

**Answer:** This implies that the algorithm had an issue parsing the image. Causes include:

- The supported input image formats includes JPEG, PNG, GIF(the first frame), BMP.
- Image file size should be no larger than 4MB
- The detectable face size range is 36x36 to 4096x4096 pixels. Faces out of this range will not be detected
- Some faces may not be detected due to technical challenges, e.g. very large face angles (head-pose), large occlusion. Frontal and near-frontal faces have the best results

# Face API Release Notes

4/18/2017 • 2 min to read • [Edit Online](#)

This article pertains to Microsoft Face API Service version 1.0.

## Release changes in March 2017

- **New Attribute** Emotion attribute could be returned from [Face - Detect](#) if `emotion` is specified in `returnFaceAttributes`.
- **Bug fix** For [Face List - Add a Face to a Face List](#) and [Person - Add a Person Face](#) APIs, algorithm is upgraded to make sure that the face could be re-detected with rectangle returned from [Face - Detect](#) as `targetFace`.
- **Bug fix** Make sure the detectable face size is strictly between 36x36 to 4096x4096 pixels.

## Release changes in November 2016

- **Face Storage Expansion;** Face Storage allows a Standard subscription to store additional persisted faces when using Person objects ([Person - Add A Person Face](#)) or Face Lists ([Face List - Add a Face to a Face List](#)) for identification or similarity matching with the Face API. The stored images are charged at \$0.5 per 1000 faces and this rate is prorated on a daily basis. Free tier subscriptions continue to be limited to 1,000 total persons.

## Release changes in October 2016

- **Error Message Change:** In [Face List Add a Face to a Face List](#) and [Person - Add a Person Face](#) APIs, the error message of more than 1 face in the targetFace changes from There are more than 1 face in the image to There is more than 1 face in the image.

## Release changes in July 2016

- **Face Verification API:** Face to Person object authentication is supported – previously Face Verification requests only supported Face to Face authentication. More details can be found here: [Face - Verify](#).
- **Finding Similar Face API:** Added optional mode field enabling selection of two working modes, default matchPerson works the same as before, and new mode matchFace removes the same person filtering. If mode field is not specified, the behavior is the same as the past release. More details can be found here: [Face - Find Similar](#).
- **Face Identification API:** Optional user-specified confidenceThreshold is enabled for user to define the confidence threshold of whether one face belong to a person object. More details can be found here: [Face - Identify](#).
- **List Person Groups:** New optional start and top parameters in list person groups to support user specifying the start point and the total person groups number to list. More details can be found here: [Person Group - List Person Groups](#).

### NOTE

All of these changes are compatible with the last version service, and using the default value for the newly added parameters will not cause any changes to users' code.

## V1.0 changes from V0

- **API Signature:** In Face API V1.0, Service root endpoint changes from <https://westus.api.cognitive.microsoft.com/face/v0/> to <https://westus.api.cognitive.microsoft.com/face/v1.0/>. There are several signature changes for API,

including:

- [Face - Detect](#)
- [Face - Identify](#)
- [Face - Find Similar](#)
- [Face - Group](#)

- **Face Sizes:** The previous version of Face API was not clear about the smallest face sizes the API could detect.

With V1.0 the API correctly sets the minimal detectable size to 36x36 pixels. Faces smaller 36x36 pixels will not be detected.

- **Persisted Data:** Existing Person Group and Person data which has been setup with Face V0 cannot be accessed with the Face V1.0 service. This incompatible issue will occur for only this one time, following API updates will not affect persisted data any more.

**NOTE**

*The V0 endpoint of Face API was retired on 06/30/2016.*

# Knowledge Exploration Service

4/12/2017 • 1 min to read • [Edit Online](#)

Welcome to the Microsoft Knowledge Exploration Service. Knowledge Exploration Service (KES) offers a fast and effective way to add interactive search and refinement to applications. With KES, you can build a compressed index from structured data, author a grammar that interprets natural language queries, and provide interactive query formulation with auto-completion suggestions.

See [Getting Started](#) for a sample walkthrough of how to use KES to create an interactive search interface for an academic publications domain.

The screenshot shows a search interface with a blue header bar. The search query is "papers about information retrieval by susan t dumais". To the right of the query is a blue search button with a white magnifying glass icon. Below the search bar, there is a list of five search results, each consisting of a snippet and a profile icon. The results are:

- papers about information retrieval by susan t dumais
- papers about information retrieval by louise t su
- papers about information retrieval by susan artandi
- papers about information retrieval by sung h myaeng
- papers about information retrieval by sung hyon myaeng

## Reference

- [Command Line Interface](#)
- [Schema Format](#)
- [Data Format](#)
- [Grammar Format](#)
- [Semantic Interpretation](#)
- [Web API Interface](#)
  - [interpret Request](#)
  - [evaluate Request](#)
  - [calchistogram Request](#)
- [Structured Query Expression](#)

# Getting Started

In this walkthrough, we will use the Knowledge Exploration Service (KES) to create the backend engine for an interactive search experience over academic publications. The command line tool [kes.exe](#) and all example files can be installed from the [Knowledge Exploration Service SDK](#).

The academic publications example contains a sample of 1000 academic papers published by researchers at Microsoft. Each paper is associated with a title, publication year, authors, and keywords. Each author is represented by an ID, name, and affiliation at the time of publication. Each keyword may be associated with a set of synonyms (ex. *support vector machine* → *svm*).

## Defining schema

The schema describes the attribute structure of the objects in our domain. It specifies the name and data type for each attribute in a JSON file format. Below is the content of the file *Academic.schema* that we use in this example:

```
{  
  "attributes": [  
    {"name": "Title", "type": "String"},  
    {"name": "Year", "type": "Int32"},  
    {"name": "Author", "type": "Composite"},  
    {"name": "Author.Id", "type": "Int64", "operations": ["equals"]},  
    {"name": "Author.Name", "type": "String"},  
    {"name": "Author.Affiliation", "type": "String"},  
    {"name": "Keyword", "type": "String", "synonyms": "Keyword.syn"}  
  ]  
}
```

Here, we define *Title*, *Year*, and *Keyword* as a string, integer, and string attribute, respectively. Since authors are represented by their ID, name, and affiliation, we define *Author* as a Composite attribute with 3 sub-attributes: *Author.Id*, *Author.Name*, and *Author.Affiliation*.

By default, attributes support all operations available for their data type, including *equals*, *starts\_with*, *is\_between*, etc. Since author ID is only used internally as an identifier, we override the default and specify *equals* as the only indexed operation.

For the *Keyword* attribute, we allow synonyms to match the canonical keyword values by specifying the synonym file *Keyword.syn* in the attribute definition. This file contains a list of canonical and synonym value pairs:

```
...  
["support vector machine", "support vector machines"]  
["support vector machine", "support vector networks"]  
["support vector machine", "support vector regression"]  
["support vector machine", "support vector"]  
["support vector machine", "svm machine learning"]  
["support vector machine", "svm"]  
["support vector machine", "svms"]  
["support vector machine", "vector machine"]  
...
```

See [Schema Format](#) for additional information about the schema definition.

## Generating data

The data file describes the list of the publications to index, with each line specifying the attribute values of a paper in [JSON format](#). Below is a single line from the data file *Academic.data*, formatted for readability:

```
...
{
    "logprob": -16.714,
    "Title": "the world wide telescope",
    "Year": 2001,
    "Author": [
        {
            "Id": 717694024,
            "Name": "alexander s szalay",
            "Affiliation": "microsoft"
        },
        {
            "Id": 2131537204,
            "Name": "jim gray",
            "Affiliation": "microsoft"
        }
    ]
}
...
```

In this snippet, we specify the *Title* and *Year* attribute of the paper as a JSON string and number, respectively. Multiple values are represented using JSON arrays. Since "Author" is a composite attribute, each value is represented using a JSON object consisting of its sub-attributes. Attributes with missing values, such as "Keyword" in this case, can be excluded from the JSON representation.

To differentiate the likelihood of different papers, we specify the relative log probability using the built-in "logprob" attribute. Given a probability  $p$  between 0 and 1, we compute the log probability as  $\log(p)$ , where  $\log()$  is the natural log function.

See [Data Format](#) for additional information about the data format.

## Building index

Once we have a schema file and data file, we can build a compressed binary index of the data objects using [`kes.exe build\_index`](#). In this example, we build the index file *Academic.index* from the input schema file *Academic.schema* and data file *Academic.data* using the following command:

```
kes.exe build_index Academic.schema Academic.data Academic.index
```

For rapid prototyping outside of Azure, [`kes.exe build\_index`](#) can build small indices locally from data files containing up to 10,000 objects. For larger data files, we can either run the command from within a [Windows VM in Azure](#), or perform a remote build in Azure. See [Scaling up](#) for details.

## Authoring grammar

The grammar specifies the set of natural language queries that the service can interpret, as well as how these natural language queries are translated into semantic query expressions. In this example, we use the grammar specified in *academic.xml*:

```
<grammar root="GetPapers">

    <!-- Import academic data schema-->
    <import schema="academic.schema" name="academic"/>
```

```

<!-- Define root rule-->
<rule id="GetPapers">
  <example>papers about machine learning by michael jordan</example>

  papers
  <tag>
    yearOnce = false;
    isBeyondEndOfQuery = false;
    query = All();
  </tag>

  <item repeat="1-" repeat-logprob="-10">
    <!-- Do not complete additional attributes beyond end of query -->
    <tag>AssertEquals(isBeyondEndOfQuery, false);</tag>

    <one-of>
      <!-- about <keyword> -->
      <item logprob="-0.5">
        about <attrref uri="academic#Keyword" name="keyword"/>
        <tag>query = And(query, keyword);</tag>
      </item>

      <!-- by <authorName> [while at <authorAffiliation>] -->
      <item logprob="-1">
        by <attrref uri="academic#Author.Name" name="authorName"/>
        <tag>authorQuery = authorName; </tag>
        <item repeat="0-1" repeat-logprob="-1.5">
          while at <attrref uri="academic#Author.Affiliation" name="authorAffiliation"/>
          <tag>authorQuery = And(authorQuery, authorAffiliation); </tag>
        </item>
        <tag>
          authorQuery = Composite(authorQuery);
          query = And(query, authorQuery);
        </tag>
      </item>
    </one-of>

    <!-- written (in|before|after) <year> -->
    <item logprob="-1.5">
      <!-- Allow this grammar path to be traversed only once -->
      <tag>
        AssertEquals(yearOnce, false);
        yearOnce = true;
      </tag>
      <ruleref uri="#GetPaperYear" name="year"/>
      <tag>query = And(query, year); </tag>
    </item>
  </one-of>

  <!-- Determine if current parse position is beyond end of query -->
  <tag>isBeyondEndOfQuery = GetVariable("IsBeyondEndOfQuery", "system"); </tag>
  </item>
  <tag>out = query; </tag>
</rule>

<rule id="GetPaperYear">
  <tag>year = All(); </tag>
  written
  <one-of>
    <item>
      in <attrref uri="academic#Year" name="year"/>
    </item>
    <item>
      before
      <one-of>
        <item>[year]</item>
        <item>
          <attrref uri="academic#Year" op="lt" name="year"/>
        </item>
      </one-of>
    </item>
  </one-of>
</rule>

```

```

</item>
<item>
  after
  <one-of>
    <item>[year]</item>
    <item>
      <attrref uri="academic#Year" op="gt" name="year"/>
    </item>
  </one-of>
</item>
</one-of>
<tag>out = year;</tag>
</rule>
</grammar>

```

For more information about the grammar specification syntax, see [Grammar Format](#).

## Compiling grammar

Once we have an XML grammar specification, we can compile it into a binary grammar using `kes.exe build_grammar`. Note that if the grammar imports a schema, the schema file needs to be located in the same path as the grammar XML. In this example, we build the binary grammar file *Academic.grammar* from the input XML grammar file *Academic.xml* using the following command:

```
kes.exe build_grammar Academic.xml Academic.grammar
```

## Hosting service

For rapid prototyping, we can host the grammar and index in a web service on the local machine using `kes.exe host_service`. Once hosted, we can access the service via [web APIs](#) to validate the data correctness and grammar design. In this example, we host the grammar file *Academic.grammar* and index file *Academic.index* at <http://localhost:8000/> using the following command:

```
kes.exe host_service Academic.grammar Academic.index --port 8000
```

This initiates a local instance of the web service. We can interactively test the service by visiting <http://localhost:<port>> from a browser. For more information, see [Testing service](#). We can also directly call various [web APIs](#) to test natural language interpretation, query completion, structured query evaluation, and histogram computation. To stop the service, enter 'quit' into the `kes.exe host_service` command prompt or press 'Ctrl+C'.

- <http://localhost:8000/interpret?query=papers by susan t dumais>
- <http://localhost:8000/interpret?query=papers by susan t d&complete=1>
- [http://localhost:8000/evaluate?expr=Composite\(Author.Name=='susan t dumais'\)&attributes=Title,Year,Author.Name,Author.Id&count=2](http://localhost:8000/evaluate?expr=Composite(Author.Name=='susan t dumais')&attributes=Title,Year,Author.Name,Author.Id&count=2)
- [http://localhost:8000/calchistogram?expr=And\(Composite\(Author.Name=='susan t dumais'\),Year>=2013\)&attributes=Year,Keyword&count=4](http://localhost:8000/calchistogram?expr=And(Composite(Author.Name=='susan t dumais'),Year>=2013)&attributes=Year,Keyword&count=4)

Outside of Azure, `kes.exe host_service` is limited to indices of up to 10,000 objects, an API rate of 10 requests per second, and a total of 1000 requests before the process automatically terminates. To bypass these restrictions, we can run the command from within a [Windows VM in Azure](#), or deploy to an Azure cloud service using the `kes.exe deploy_service` command. See [Deploying service](#) for details.

## Scaling up

When running `kes.exe` outside of Azure, the index is limited to 10,000 objects. Once we are ready to scale up, we can build and host larger indices using Azure. We can sign up for a [free trial](#). Alternatively, for Visual Studio/MSDN

subscriber, we can [activate subscriber benefits](#) which offer some Azure credits each month.

To allow `kes.exe` access to an Azure account, visit <https://manage.windowsazure.com/publishsettings/> to download the Azure Publish Settings file. If prompted, sign into the desired Azure account. Once signed in, the browser will automatically download the Publish Settings file. Save it as `AzurePublishSettings.xml` in the working directory from where `kes.exe` runs.

There are two ways to build and host large indices. The first is to prepare the schema and data files in a Windows VM in Azure and run `kes.exe build_index` to build the index locally on the VM without any size restrictions. The resulting index can be hosted locally on the VM using `kes.exe host_service` for rapid prototyping, again without any restrictions. See the [Azure VM tutorial](#) for detailed steps to create an Azure VM.

The second method is to perform a remote Azure build using `kes.exe build_index` with the `--remote` parameter, which specifies an Azure VM size. When the `--remote` parameter is specified, the command creates a temporary Azure VM of that size, builds the index on the VM, uploads the index to the target blob storage, and deletes the VM upon completion. Your Azure subscription is charged for the cost of the VM while the index is being built. This remote Azure build capability allows `kes.exe build_index` to be executed in any environment. When performing a remote build, the input schema and data arguments may be local file paths or [Azure blob storage](#) URLs. The output index argument must be a blob storage URL. To create an Azure storage account, see [Create Storage Account](#). Note that only classic storage accounts are supported at this time. We can use the utility [AzCopy](#) to efficiently copy files to and from blob storage.

In this example, we assume that the blob storage container `http://<account>.blob.core.windows.net/<container>` has already been created, containing the schema `Academic.schema`, referenced synonym file `Keywords.syn`, and full-scale data file `Academic.full.data`. We can build the full index remotely using the following command:

```
kes.exe build_index http://<account>.blob.core.windows.net/<container>/Academic.schema  
http://<account>.blob.core.windows.net/<container>/Academic.full.data  
http://<account>.blob.core.windows.net/<container>/Academic.full.index --remote Large
```

Note that it may take 5-10 minutes to provision a temporary VM to build the index. Thus, for rapid prototyping, we recommend one of two options:

1. Develop with a smaller data set locally on any machine.
2. Manually [create an Azure VM, connect to it](#) via Remote Desktop, install the [Knowledge Exploration Service SDK](#), and run `kes.exe` from within the VM.

To avoid paging which slows down the build process, we recommend using a VM with 3 times the amount of RAM as the input data file size for index building, and a VM with 1 GB more RAM than the index size for hosting. For a list of available VM sizes, see [Sizes for virtual machines](#).

## Deploying service

Once we have a grammar and index, we are ready to deploy the service to an Azure cloud service. To create a new Azure cloud service, see [How to Create and Deploy a Cloud Service](#). Do not specify a deployment package at this point.

Once the cloud service has been created, we can use `kes.exe deploy_service` to deploy the service. An Azure cloud service has two deployment slots: Production and Staging. For a service that receives live user traffic, we should initially deploy to the Staging slot and wait for the service to start up and initialize itself. Once the service is running, we can send a few requests to validate the deployment and verify that it passes basic tests. Then, we [swap](#) the contents of the Staging slot with the Production slot so that live traffic will now be directed to the newly deployed service. We can repeat this process when deploying an updated version of the service with new data. Like all other Azure cloud services, we can optionally use the Azure portal to configure [auto-scaling](#).

In this example, we will deploy the Academic index to the staging slot of an existing cloud service with *large* VMs using the following command:

```
kes.exe deploy_service http://<account>.blob.core.windows.net/<container>/Academic.grammar  
http://<account>.blob.core.windows.net/<container>/Academic.index <serviceName> large --slot Staging
```

For a list of available VM sizes, see [Sizes for virtual machines](#). Once the service has been deployed, we can call the various [web APIs](#) to test natural language interpretation, query completion, structured query evaluation, and histogram computation.

## Testing service

To debug a live service, we can simply navigate to the host machine from a web browser. For a local service deployed via [host\\_service](#), visit `http://localhost:<port>/`. For an Azure cloud service deployed via [deploy\\_service](#), visit `http://<serviceName>.cloudapp.net/`. This page contains a link to information about basic API call statistics as well as the grammar and index hosted at this service. This page also contains an interactive search interface that demonstrates the use of the web APIs. We can enter queries interactively into the search box to see the results of the [interpret](#), [evaluate](#), and [calchistogram](#) API calls. The underlying HTML source of this page also serves as an example of how to integrate the web APIs into an app to create a rich interactive search experience.

# Command Line Interface

4/12/2017 • 4 min to read • [Edit Online](#)

The KES command line interface provides the ability to build index and grammar files from structured data and deploy them as web services. It uses the general syntax: `kes.exe <command> <required_args> [<optional_args>]`. You can run `kes.exe` without arguments to display a list of commands, or `kes.exe <command>` to display a list of arguments available for the specified command. Below is a list of available commands:

- `build_index`
- `build_grammar`
- `host_service`
- `deploy_service`
- `describe_index`
- `describe_grammar`

## `build_index` Command

The **build\_index** command builds a binary index file from a schema definition file and a data file of objects to be indexed. The resulting index file can be used to evaluate structured query expressions, or to generate interpretations of natural language queries in conjunction with a compiled grammar file.

```
kes.exe build_index <schemaFile> <dataFile> <indexFile> [options]
```

PARAMETER	DESCRIPTION
<code>&lt;schemaFile&gt;</code>	Input schema path
<code>&lt;dataFile&gt;</code>	Input data path
<code>&lt;indexFile&gt;</code>	Output index path
<code>--description &lt;description&gt;</code>	Description string
<code>--remote &lt;vmSize&gt;</code>	Size of VM for remote build

These files may be specified by local file paths or URL paths to Azure blobs. The schema file describes the structure of the objects being indexed as well as the operations to be supported (see [Schema Format](#)). The data file enumerates the objects and attribute values to be indexed (see [Data Format](#)). When the build succeeds, the output index file contains a compressed representation of the input data that supports the desired operations.

A description string may be optionally specified to subsequently identify a binary index using the **describe\_index** command.

By default, the index is built on the local machine. Outside of the Azure environment, local builds are limited to data files containing up to 10,000 objects. When the --remote flag is specified, the index will be built on a temporarily created Azure VM of the specified size. This allows large indices to be built efficiently using Azure VMs with more memory. To avoid paging which slows down the build process, we recommend using a VM with 3 times the amount of RAM as the input data file size. For a list of available VM sizes, see [Sizes for virtual machines](#).

## TIP

For faster builds, presort the objects in the data file by decreasing probability.

## build\_grammar Command

The **build\_grammar** command compiles a grammar specified in XML to a binary grammar file. The resulting grammar file can be used in conjunction with an index file to generate interpretations of natural language queries.

```
kes.exe build_grammar <xmlFile> <grammarFile>
```

PARAMETER	DESCRIPTION
<xmlFile>	Input XML grammar specification path
<grammarFile>	Output compiled grammar path

These files may be specified by local file paths or URL paths to Azure blobs. The grammar specification describes the set of weighted natural language expressions and their semantic interpretations (see [Grammar Format](#)). When the build succeeds, the output grammar file contains a binary representation of the grammar specification to enable fast decoding.

## host\_service Command

The **host\_service** command hosts an instance of the KES service on the local machine.

```
kes.exe host_service <grammarFile> <indexFile> [options]
```

PARAMETER	DESCRIPTION
<grammarFile>	Input binary grammar path
<indexFile>	Input binary index path
--port <port>	Local port number. Default: 8000

These files may be specified by local file paths or URL paths to Azure blobs. A web service will be hosted at <http://localhost:<port>/>. See [Web APIs](#) for a list of supported operations.

Outside of the Azure environment, locally hosted services are limited to index files up to 1 MB in size, 10 requests per second, and 1000 total calls. To overcome these limitations, run **host\_service** inside an Azure VM, or deploy to an Azure cloud service using **deploy\_service**.

## deploy\_service Command

The **deploy\_service** command deploys an instance of the KES service to an Azure cloud service.

```
kes.exe deploy_service <grammarFile> <indexFile> <serviceName> <vmSize>[options]
```

PARAMETER	DESCRIPTION
<grammarFile>	Input binary grammar path

PARAMETER	DESCRIPTION
<indexFile>	Input binary index path
<serviceName>	Name of target cloud service
<vmSize>	Size of cloud service VM
--slot <slot>	Cloud service slot: "staging" (default), "production"

These files may be specified by local file paths or URL paths to Azure blobs. Service name specifies a preconfigured Azure cloud service (see [How to Create and Deploy a Cloud Service](#)). The command will automatically deploy the KES service to the specified Azure cloud service, using VMs of the specified size. To avoid paging which significantly decreases performance, we recommend using a VM with 1 GB more RAM than the input index file size. For a list of available VM sizes, see [Sizes for Cloud Services](#).

By default, the service is deployed to the staging environment, optionally overridden via the --slot parameter. See [Web APIs](#) for a list of supported operations.

## describe\_index command

The **describe\_index** command outputs information about an index file, including the schema and description.

```
kes.exe describe_index <indexFile>
```

PARAMETER	DESCRIPTION
<indexFile>	Input index path

This file may be specified by a local file path or a URL path to an Azure blob. The output description string can be specified using the --description parameter of the **build\_index** command.

## describe\_grammar command

The **describe\_grammar** command outputs the original grammar specification used to build the binary grammar.

```
kes.exe describe_grammar <grammarFile>
```

PARAMETER	DESCRIPTION
<grammarFile>	Input grammar path

This file may be specified by a local file path or a URL path to an Azure blob.

# Schema Format

4/12/2017 • 3 min to read • [Edit Online](#)

The schema is specified in a JSON file that describes the attribute structure of the objects in the data file used to create the index. For each attribute, the schema specifies the name, data type, optional operations, and optional synonyms list. An object may have 0 or more values of each attribute. Below is a simplified example from an academic publication domain:

```
{  
  "attributes": [  
    {"name": "Title", "type": "String"},  
    {"name": "Year", "type": "Int32"},  
    {"name": "Author", "type": "Composite"},  
    {"name": "Author.Id", "type": "Int64", "operations": ["equals"]},  
    {"name": "Author.Name", "type": "String"},  
    {"name": "Author.Affiliation", "type": "String"},  
    {"name": "Keyword", "type": "String", "synonyms": "Keyword.syn"}  
  ]  
}
```

Attribute names are case-sensitive identifiers that start with a letter and consist only of letters (A-Z), numbers (0-9), and underscore (\_). The reserved "logprob" attribute is used to specify the relative natural log probabilities among objects.

## Attribute Type

Below is a list of supported attribute data types:

TYPE	DESCRIPTION	OPERATIONS	EXAMPLE
String	String (1-1024 characters)	equals, starts_with	"hello world"
Int32	Signed 32-bit integer	equals, starts_with, is_between	2016
Int64	Signed 64-bit integer	equals, starts_with, is_between	9876543210
Double	Double-precision floating-point value	equals, starts_with, is_between	1.602e-19
Date	Date (1400-01-01 to 9999-12-31)	equals, is_between	'2016-03-14'
Guid	Globally unique identifier	equals	"602DD052-CC47-4B23-A16A-26B52D30C05B"
Blob	Internally compressed non-indexed data	None	"Empower every person and every organization on the planet to achieve more"

Type	Description	Operations	Example
Composite	Composition of multiple sub-attributes	N/A	{ "Name":"harry shum", "Affiliation":"microsoft" }

String attributes are used to represent string values that may appear as part of the user query. They support the exact-match *equals* operation, as well as the *starts\_with* operation for query completion scenarios, such as matching "micros" with "microsoft". Case-insensitive and fuzzy matching to handle spelling errors will be supported in a future release.

Int32/Int64/Double attributes are used to represent numeric values. The *is\_between* operation enables inequality support (lt, le, gt, ge) at run time. The *starts\_with* operation supports query completion scenarios, such as matching "20" with "2016", or "3." with "3.14".

Date attributes are used to efficiently encode date values. The *is\_between* operation enables inequality support (lt, le, gt, ge) at run time.

Guid attributes are used to efficiently represent GUID values with default support for the *equals* operation.

Blob attributes are used to efficiently encode potentially large data blobs for runtime lookup from the corresponding object, without support for any indexing operation based on the content of the blob values.

### Composite Attributes

Composite attributes are used to represent a grouping of attribute values. The name of each sub-attribute starts with the name of the composite attribute followed by ". ". Values for composite attributes are specified as a JSON object containing the nested attribute values. Composite attributes may have multiple object values. However, composite attributes may not have sub-attributes that are themselves composite attributes.

In the academic publication example above, this enables the service to query for papers by "harry shum" while he is at "microsoft". Without composite attributes, the service can only query for papers where one of the authors is "harry shum" and one of the authors is at "microsoft". For more information, see [Composite Queries](#).

## Attribute Operations

By default, each attribute is indexed to support all operations available to the attribute data type. If a particular operation is not required, the set of indexed operations can be explicitly specified to reduce the size of the index. In the following snippet from the example schema above, the Author.Id attribute is indexed to support only the *equals* operation, but not the additional *starts\_with* and *is\_between* operations for Int32 attributes.

```
{"name": "Author.Id", "type": "Int32", "operations": ["equals"]}
```

When an attribute is referenced inside a grammar, the *starts\_with* operation needs to be specified in the schema in order for the service to generate completions from partial queries.

## Attribute Synonyms

It is often desirable to refer to a particular string attribute value by a synonym. For example, users may refer to "Microsoft" as "MSFT" or "MS". In these cases, the attribute definition can specify the name of a schema file located in the same directory as the schema file. Each line in the synonym file represents a synonym entry in the following JSON format: [ "<canonical>", "<synonym>" ]. In the example schema, "AuthorName.syn" is a JSON file that contains synonym values for the Author.Name attribute.

```
{"name": "Author.Name", "type": "String", "synonyms": "AuthorName.syn"}
```

A canonical value may have multiple synonyms. Multiple canonical values may share a common synonym. In such

cases, the service will resolve the ambiguity through multiple interpretations. Below is an example AuthorName.syn synonyms file corresponding to the schema above:

```
[ "harry shum", "heung-yeung shum" ]  
[ "harry shum", "h shum" ]  
[ "henry shum", "h shum" ]  
...
```

# Data Format

4/12/2017 • 1 min to read • [Edit Online](#)

The data file describes the list of objects to index. Each line in the file specifies the attribute values of an object in [JSON format](#) with UTF-8 encoding. In addition to the attributes defined in the [schema](#), each object has an optional "logprob" attribute that specifies the relative log probability among the objects. When the service returns objects in order of decreasing probability, we can use "logprob" to indicate the return order of matching objects. Given a probability  $p$  between 0 and 1, the corresponding log probability can be computed as  $\log(p)$ , where  $\log()$  is the natural log function. When no value is specified for logprob, the default value 0 is used.

```
{"logprob": -5.3, "Title": "latent dirichlet allocation", "Year": 2003, "Author": {"Name": "david m blei", "Affiliation": "uc berkeley"}, "Author": {"Name": "andrew y ng", "Affiliation": "stanford"}, "Author": {"Name": "michael i jordan", "Affiliation": "uc berkeley"}}, {"logprob": -6.1, "Title": "probabilistic latent semantic indexing", "Year": 1999, "Author": {"Name": "thomas hofmann", "Affiliation": "uc berkeley"}}, ...
```

For String, GUID, and Blob attributes, the value is represented as a quoted JSON string. For numeric attributes (Int32, Int64, Double), the value is represented as a JSON number. For composite attributes, the value is a JSON object that specifies the sub-attribute values. For faster index builds, presort the objects by decreasing log probability.

In general, an attribute may have 0 or more values. If an attribute has no value, we simply drop it from the JSON. If an attribute has 2 or more values, we can repeat the attribute value pair in the JSON object. Alternatively, we can assign a JSON array containing the multiple values to the attribute.

```
{"logprob": 0, "Title": "0 keyword"}, {"logprob": 0, "Title": "1 keyword", "Keyword": "foo"}, {"logprob": 0, "Title": "2 keywords", "Keyword": "foo", "Keyword": "bar"}, {"logprob": 0, "Title": "2 keywords (alt)", "Keyword": ["foo", "bar"]}
```

# Grammar Format

4/12/2017 • 8 min to read • [Edit Online](#)

The grammar is an XML file that specifies the weighted set of natural language queries that the service can interpret, as well as how these natural language queries are translated into semantic query expressions. The grammar syntax is based on [SRGS](#), a W3C standard for speech recognition grammars, with extensions to support data index integration and semantic functions.

The following describes each of the syntactic elements that can be used in a grammar. See [this example](#) for a complete grammar that demonstrates the use of these elements in context.

## grammar Element

The `grammar` element is the top-level element in the grammar specification XML. The required `root` attribute specifies the name of the root rule that defines the starting point of the grammar.

```
<grammar root="GetPapers">
```

## import Element

The `import` element imports a schema definition from an external file to enable attribute references. The element must be a child of the top-level `grammar` element and appear before any `attrref` elements. The required `schema` attribute specifies the name of a schema file located in the same directory as the grammar XML file. The required `name` element specifies the schema alias that subsequent `attrref` elements use when referencing attributes defined within this schema.

```
<import schema="academic.schema" name="academic"/>
```

## rule Element

The `rule` element defines a grammar rule, a structural unit that specifies a set of query expressions that the system can interpret. The element must be a child of the top-level `grammar` element. The required `id` attribute specifies the name of the rule, which is referenced from `grammar` or `ruleref` elements.

A `rule` element defines a set of legal expansions. Text tokens match against the input query directly. `item` elements specify repeats and alter interpretation probabilities. `one-of` elements indicate alternative choices. `ruleref` elements enable construction of more complex expansions from simpler ones. `attrref` elements allow matches against attribute values from the index. `tag` elements specify the semantics of the interpretation and can alter the interpretation probability.

```
<rule id="GetPapers">....</rule>
```

## example Element

The optional `example` element specifies example phrases that may be accepted by the containing `rule` definition. This may be used for documentation and/or automated testing.

```
<example>papers about machine learning by michael jordan</example>
```

## item Element

The `item` element groups a sequence of grammar constructs. It can be used to indicate repetitions of the expansion sequence, or to specify alternatives in conjunction with the `one-of` element.

When an `item` element is not a child of a `one-of` element, it can specify repetition of the enclosed sequence by assigning the `repeat` attribute to a count value. A count value of " $n$ " (where  $n$  is an integer) indicates that the sequence must occur exactly  $n$  times. A count value of " $m-n$ " allows the sequence to appear between  $m$  and  $n$  times, inclusively. A count value of " $m-$ " specifies that the sequence must appear at least  $m$  times. The optional `repeat-logprob` attribute can be used to alter the interpretation probability for each additional repetition beyond the minimum.

```
<item repeat="1-" repeat-logprob="-10">...</item>
```

When `item` elements appear as children of a `one-of` element, they define the set of expansion alternatives. In this usage, the optional `logprob` attribute specifies the relative log probability among the different choices. Given a probability  $p$  between 0 and 1, the corresponding log probability can be computed as  $\log(p)$ , where  $\log()$  is the natural log function. If not specified, `logprob` defaults to 0, which does not alter the interpretation probability. Note that log probability is always a negative floating-point value or 0.

```
<one-of>
  <item>by</item>
  <item logprob="-0.5">written by</item>
  <item logprob="-1">authored by</item>
</one-of>
```

## one-of Element

The `one-of` element specifies alternative expansions among one of the child `item` elements. Only `item` elements may appear inside a `one-of` element. Relative probabilities among the different choices may be specified via the `logprob` attribute in each child `item`.

```
<one-of>
  <item>by</item>
  <item logprob="-0.5">written by</item>
  <item logprob="-1">authored by</item>
</one-of>
```

## ruleref Element

The `ruleref` element specifies valid expansions via references to another `rule` element. Through the use of `ruleref` elements, more complex expressions can be built from simpler rules. The required `uri` attribute indicates the name of the referenced `rule` using the syntax "#ruleName". To capture the semantic output of the referenced rule, use the optional `name` attribute to specify the name of a variable to which the semantic output is assigned.

```
<ruleref uri="#GetPaperYear" name="year"/>
```

## attrref Element

The `attrref` element references an index attribute, allowing matching against attribute values observed in the index. The required `uri` attribute specifies the index schema name and attribute name using the syntax "schemaName#attrName". There must be a preceding `import` element that imports the schema named `schemaName`. The attribute name is the name of an attribute defined in the corresponding schema.

In addition to matching user input, the `attrref` element also returns a structured query object as output that

selects the subset of objects in the index matching the input value. Use the optional `name` attribute to specify the name of the variable where the query object output should be stored. The query object can be composed with other query objects to form more complex expressions. See [Semantic Interpretation](#) for details.

```
<attrref uri="academic#Keyword" name="keyword"/>
```

### Query Completion

To support query completions when interpreting partial user queries, each referenced attribute must include "starts\_with" as an operation in the schema definition. Given a user query prefix, `attrref` will match all values in the index that complete the prefix, and yield each complete value as a separate interpretation of the grammar.

Examples:

- Matching `<attrref uri="academic#Keyword" name="keyword"/>` against the query prefix "dat" generates one interpretation for papers about "database", one interpretation for papers about "data mining", etc.
- Matching `<attrref uri="academic#Year" name="year"/>` against the query prefix "200" generates one interpretation for papers in "2000", one interpretation for papers in "2001", etc.

### Matching Operations

In addition to exact match, select attribute types also support prefix and inequality matches via the optional `op` attribute. If no object in the index has a value that matches, the grammar path is blocked and the service will not generate any interpretations traversing over this grammar path. The `op` attribute defaults to "eq".

```
in <attrref uri="academic#Year" name="year"/>
before <attrref uri="academic#Year" op="lt" name="year"/>
```

The following table lists the supported `op` values for each attribute type. Their use requires the corresponding index operation to be included in the schema attribute definition.

ATTRIBUTE TYPE	OP VALUE	DESCRIPTION	INDEX OPERATION
String	eq	String exact match	equals
String	starts_with	String prefix match	starts_with
Int32, Int64, Double	eq	Numeric equality match	equals
Int32, Int64, Double	lt, le, gt, ge	Numeric inequality match (<, <=, >, >=)	is_between
Int32, Int64, Double	starts_with	Prefix match of value in decimal notation	starts_with

Examples:

- `<attrref uri="academic#Year" op="lt" name="year"/>` matches the input string "2000" and returns all papers published before the year 2000, exclusively.
- `<attrref uri="academic#Year" op="lt" name="year"/>` does not match the input string "20" because there are no papers in the index published before the year 20.
- `<attrref uri="academic#Keyword" op="starts_with" name="keyword"/>` matches the input string "dat" and returns in a single interpretation papers about "database", "data mining", etc. This is a rare use case.
- `<attrref uri="academic#Year" op="starts_with" name="year"/>` matches the input string "20" and returns in a single interpretation papers published in 200-299, 2000-2999, etc. This is a rare use case.

## tag Element

The `tag` element specifies how a path through the grammar is to be interpreted. It contains a sequence of semicolon-terminated statements. A statement may be an assignment of a literal or a variable to another variable. It may also assign the output of a function with 0 or more parameters to a variable. Each function parameter may be specified using a literal or a variable. If the function does not return any output, the assignment is omitted. Variable scope is local to the containing rule.

```
<tag>x = 1; y = x;</tag>
<tag>q = All(); q = And(q, q2);</tag>
<tag>AssertEquals(x, 1);</tag>
```

Each `rule` in the grammar has a predefined variable named "out", representing the semantic output of the rule. Its value is computed by evaluating each of the semantic statements traversed by the path through the `rule` matching the user query input. The value assigned to the "out" variable at the end of the evaluation is the semantic output of the rule. The semantic output of interpreting a user query against the grammar is the semantic output of the root rule.

Some statements may alter the probability of an interpretation path by introducing an additive log probability offset. Some statements may reject the interpretation altogether if specified conditions are not satisfied.

For a list of supported semantic functions, see [Semantic Functions](#).

## Interpretation Probability

The probability of an interpretation path through the grammar is the cumulative log probability of all the `<item>` elements and semantic functions encountered along the way. It describes the relative likelihood of matching a particular input sequence.

Given a probability  $p$  between 0 and 1, the corresponding log probability can be computed as  $\log(p)$ , where  $\log()$  is the natural log function. Using log probabilities allows the system to accumulate the joint probability of an interpretation path through simple addition. It also avoids floating-point underflow common to such joint probability calculations. Note that by design, the log probability is always a negative floating-point value or 0, where larger values indicate higher likelihood.

## Example

The following is an example XML from the academic publications domain that demonstrates the various elements of a grammar:

```
<grammar root="GetPapers">

    <!-- Import academic data schema-->
    <import schema="academic.schema" name="academic"/>

    <!-- Define root rule-->
    <rule id="GetPapers">
        <example>papers about machine learning by michael jordan</example>

        papers
        <tag>
            yearOnce = false;
            isBeyondEndOfQuery = false;
            query = All();
        </tag>

        <item repeat="1-" repeat-logprob="-10">
            <!-- Do not complete additional attributes beyond end of query -->
            <tag>AssertEquals(isBeyondEndOfQuery, false);</tag>
        </item>
    </rule>
</grammar>
```

```

<one-of>
    <!-- about <keyword> -->
    <item logprob="-0.5">
        about <attrref uri="academic#Keyword" name="keyword"/>
        <tag>query = And(query, keyword);</tag>
    </item>

    <!-- by <authorName> [while at <authorAffiliation>] -->
    <item logprob="-1">
        by <attrref uri="academic#Author.Name" name="authorName"/>
        <tag>authorQuery = authorName;</tag>
        <item repeat="0-1" repeat-logprob="-1.5">
            while at <attrref uri="academic#Author.Affiliation" name="authorAffiliation"/>
            <tag>authorQuery = And(authorQuery, authorAffiliation);</tag>
        </item>
        <tag>
            authorQuery = Composite(authorQuery);
            query = And(query, authorQuery);
        </tag>
    </item>

    <!-- written (in|before|after) <year> -->
    <item logprob="-1.5">
        <!-- Allow this grammar path to be traversed only once -->
        <tag>
            AssertEquals(yearOnce, false);
            yearOnce = true;
        </tag>
        <ruleref uri="#GetPaperYear" name="year"/>
        <tag>query = And(query, year);</tag>
    </item>
</one-of>

<!-- Determine if current parse position is beyond end of query -->
<tag>isBeyondEndOfQuery = GetVariable("IsBeyondEndOfQuery", "system");</tag>
</item>
<tag>out = query;</tag>
</rule>

<rule id="GetPaperYear">
    <tag>year = All();</tag>
    written
    <one-of>
        <item>
            in <attrref uri="academic#Year" name="year"/>
        </item>
        <item>
            before
            <one-of>
                <item>[year]</item>
                <item><attrref uri="academic#Year" op="lt" name="year"/></item>
            </one-of>
        </item>
        <item>
            after
            <one-of>
                <item>[year]</item>
                <item><attrref uri="academic#Year" op="gt" name="year"/></item>
            </one-of>
        </item>
    </one-of>
    <tag>out = year;</tag>
</rule>
</grammar>

```

# Semantic Interpretation

4/12/2017 • 4 min to read • [Edit Online](#)

Semantic interpretation associates semantic output with each interpreted path through the grammar. In particular, the service evaluates the sequence of statements in the `tag` elements traversed by the interpretation to compute the final output.

A statement may be an assignment of a literal or a variable to another variable. It may also assign the output of a function with 0 or more parameters to a variable. Each function parameter may be specified using a literal or a variable. If the function does not return any output, the assignment is omitted.

```
<tag>x = 1; y = x;</tag>
<tag>q = All(); q = And(q, q2);</tag>
<tag>AssertEquals(x, 1);</tag>
```

A variable is specified using a name identifier that starts with a letter and consists only of letters (A-Z), numbers (0-9), and the underscore (\_). Its type is implicitly inferred from the literal or function output value assigned to it.

Below is a list of currently supported data types:

TYPE	DESCRIPTION	EXAMPLES
String	Sequence of 0 or more characters	"Hello World!" ""
Bool	Boolean value	true false
Int32	32-bit signed integer. -2.1e9 to 2.1e9	123 -321
Int64	64-bit signed integer. -9.2e18 and 9.2e18	9876543210
Double	Double precision floating-point. 1.7e+/- 308 (15 digits)	123.456789 1.23456789e2
Guid	Globally unique identifier	"602DD052-CC47-4B23-A16A-26B52D30C05B"
Query	Query expression that specifies a subset of data objects in the index	All() And(q1, q2)

## Semantic Functions

There is a built-in set of semantic functions. They allow the construction of sophisticated queries, and provide context sensitive control over grammar interpretations.

### And Function

```
query = And(query1, query2);
```

Returns a query composed from the intersection of two input queries.

## Or Function

```
query = Or(query1, query2);
```

Returns a query composed from the union of two input queries.

## All Function

```
query = All();
```

Returns a query that includes all data objects.

In the following example, we use the All() function to iteratively build up a query based on the intersection of 1 or more keywords.

```
<tag>query = All();</tag>
<item repeat="1-"
  <attrref uri="academic#Keyword" name="keyword">
    <tag>query = And(query, keyword);</tag>
  </item>
```

## None Function

```
query = None();
```

Returns a query that includes no data objects.

In the following example, we use the None() function to iteratively build up a query based on the union of 1 or more keywords.

```
<tag>query = None();</tag>
<item repeat="1-"
  <attrref uri="academic#Keyword" name="keyword">
    <tag>query = Or(query, keyword);</tag>
  </item>
```

## Query Function

```
query = Query(attrName, value)
query = Query(attrName, value, op)
```

Returns a query that includes only data objects whose attribute *attrName* matches value *value* according to the specified operation *op*, which defaults to "eq". Typically, use an `attrref` element to create a query based on the matched input query string. If a value is given or obtained through other means, the `Query()` function can be used to create a query matching this value.

In the following example, we use the `Query()` function to implement support for specifying academic publications from a particular decade.

```
written in the 90s
<tag>
  beginYear = Query("academic#Year", 1990, "ge");
  endYear = Query("academic#Year", 2000, "lt");
  year = And(beginYear, endYear);
</tag>
```

## Composite Function

```
query = Composite(innerQuery);
```

Returns a query that encapsulates an *innerQuery* composed of matches against sub-attributes of a common composite attribute *attr*. The encapsulation requires the composite attribute *attr* of any matching data object to have at least one value that individually satisfies the *innerQuery*. Note that a query on sub-attributes of a composite attribute has to be encapsulated using the `Composite()` function before it can be combined with other queries.

For example, the following query returns academic publications by "harry shum" while he was at "microsoft":

```
Composite(And(Query("academic#Author.Name", "harry shum"),
    Query("academic#Author.Affiliation", "microsoft")));
```

On the other hand, the following query returns academic publications where one of the authors is "harry shum" and one of the affiliations is "microsoft":

```
And(Composite(Query("academic#Author.Name", "harry shum"),
    Composite(Query("academic#Author.Affiliation", "microsoft"))));
```

## GetVariable Function

```
value = GetVariable(name, scope);
```

Returns the value of variable *name* defined under the specified *scope*. *name* is an identifier that starts with a letter and consists only of letters (A-Z), numbers (0-9), and the underscore (\_). *scope* can be set to "request" or "system". Note that variables defined under different scopes are distinct from each other, including ones defined via the output of semantic functions.

Request scope variables are shared across all interpretations within the current interpret request. They can be used to control the search for interpretations over the grammar.

System variables are predefined by the service and can be used to retrieve various statistics about the current state of the system. Below is the set of currently supported system variables:

NAME	TYPE	DESCRIPTION
IsAtEndOfQuery	Bool	true if the current interpretation has matched all input query text
IsBeyondEndOfQuery	Bool	true if the current interpretation has suggested completions beyond the input query text

## SetVariable Function

```
SetVariable(name, value, scope);
```

Assigns *value* to variable *name* under the specified *scope*. *name* is an identifier that starts with a letter and consists only of letters (A-Z), numbers (0-9), and the underscore (\_). Currently, the only valid value for *scope* is "request". There are no settable system variables.

Request scope variables are shared across all interpretations within the current interpret request. They can be used to control the search for interpretations over the grammar.

## AssertEquals Function

```
AssertEquals(value1, value2);
```

If *value1* and *value2* are equivalent, the function succeeds and has no side effects. Otherwise, the function fails and rejects the interpretation.

## **AssertNotEquals Function**

```
AssertNotEquals(value1, value2);
```

If *value1* and *value2* are not equivalent, the function succeeds and has no side effects. Otherwise, the function fails and rejects the interpretation.

# Web API Interface

4/12/2017 • 1 min to read • [Edit Online](#)

The model files built by the Knowledge Exploration Service can be hosted and accessed via a set of web APIs. The APIs may be hosted on the local machine using the `host_service` command, or may be deployed to an Azure cloud service using the `deploy_service` command. Both techniques expose the following API endpoints:

- *interpret* – Interprets a natural language query string. Returns annotated interpretations to enable rich search-box auto-completion experiences that anticipate what the user is typing.
- *evaluate* – Evaluates and returns the output of a structured query expression.
- *calchistogram* – Calculates a histogram of attribute values for objects returned by a structured query expression.

Used together, these API methods allow the creation of a rich semantic search experience. Given a natural language query string, the *interpret* method provides annotated versions of the input query with structured query expressions, based on the underlying grammar and index data. The *evaluate* method evaluates the structured query expression and returns the matching index objects for display. The *calchistogram* method computes the attribute value distributions to enable filtering and refinement.

## Example

In an academic publications domain, if a user types the string "latent s", the *interpret* method can provide a set of ranked interpretations, suggesting that the user might be searching for the keyword "latent semantic analysis", the title "latent structure analysis", or other expressions starting with "latent s". This information can be used to quickly guide the user to the desired search results.

For this domain, the *evaluate* method can be used to retrieve a set of matching publications from the academic index, and the *calchistogram* method can be used to calculate the distribution of attribute values for the matching publications, which can be used to further filter and refine the search results.

Note that to improve the readability of the examples, the REST API calls contain characters (such as spaces) that have not been URL-encoded. Your code will need to apply the appropriate URL-encodings.

# calchistogram Method

4/12/2017 • 1 min to read • [Edit Online](#)

The *calchistogram* method computes the objects matching a structured query expression and calculates the distribution of their attribute values.

## Request

`http://<host>/calchistogram?expr=<expr>[&options]`

NAME	VALUE	DESCRIPTION
expr	Text string	Structured query expression that specifies the index entities over which to calculate histograms.
attributes	Text string (default="")	Comma-delimited list of attribute to included in the response.
count	Number (default=10)	Number of results to return.
offset	Number (default=0)	Index of the first result to return.

## Response (JSON)

JSONPATH	DESCRIPTION
<code>\$.expr</code>	<i>expr</i> parameter from the request.
<code>\$.num_entities</code>	Total number of matching entities.
<code>\$.histograms</code>	Array of histograms, one for each requested attribute.
<code>\$.histograms[*].attribute</code>	Name of the attribute over which the histogram was computed.
<code>\$.histograms[*].distinct_values</code>	Number of distinct values among matching entities for this attribute.
<code>\$.histograms[*].total_count</code>	Total number of value instances among matching entities for this attribute.
<code>\$.histograms[*].histogram</code>	Histogram data for this attribute.
<code>\$.histograms[*].histogram[*].value</code>	Attribute value.
<code>\$.histograms[*].histogram[*].logprob</code>	Total natural log probability of matching entities with this attribute value.
<code>\$.histograms[*].histogram[*].count</code>	Number of matching entities with this attribute value.

JSONPATH	DESCRIPTION
\$.aborted	True if the request timed out.

### Example

In the academic publications example, the following calculates a histogram of publication counts by year and by keyword for a particular author since 2013:

```
http://<host>/calchistogram?expr=And(Composite(Author.Name=='jaime
teeven'),Year>=2013)&attributes=Year,Keyword&count=4
```

The response indicates that there are 37 papers matching the query expression. For the *Year* attribute, there are 3 distinct values, one for each year since 2013. The total paper count over the 3 distinct values is 37. For each *Year*, the histogram shows the value, total natural log probability, and count of matching entities.

The histogram for *Keyword* shows that there are 34 distinct keywords. As a paper may be associated with multiple keywords, the total count (53) can be larger than the number of matching entities. Although there are 34 distinct values, the response only includes the top 4 because of the "count=4" parameter.

```
{  
  "expr": "And(Composite(Author.Name=='jaime teeven'),Y>=2013)",  
  "num_entities": 37,  
  "histograms": [  
    {  
      "attribute": "Y",  
      "distinct_values": 3,  
      "total_count": 37,  
      "histogram": [  
        {  
          "value": 2014,  
          "logprob": -6.894,  
          "count": 15  
        },  
        {  
          "value": 2013,  
          "logprob": -6.927,  
          "count": 12  
        },  
        {  
          "value": 2015,  
          "logprob": -7.082,  
          "count": 10  
        }  
      ]  
    },  
    {  
      "attribute": "Keyword",  
      "distinct_values": 34,  
      "total_count": 53,  
      "histogram": [  
        {  
          "value": "crowdsourcing",  
          "logprob": -7.142,  
          "count": 9  
        },  
        {  
          "value": "information retrieval",  
          "logprob": -7.389,  
          "count": 4  
        },  
        {  
          "value": "personalization",  
          "logprob": -7.623,  
          "count": 3  
        },  
        {  
          "value": "mobile search",  
          "logprob": -7.674,  
          "count": 2  
        }  
      ]  
    }  
  ]  
}
```

# evaluate Method

4/12/2017 • 1 min to read • [Edit Online](#)

The *evaluate* method evaluates and returns the output of a structured query expression based on the index data.

Typically, an expression will be obtained from a response to the *interpret* method. But you can also compose query expressions yourself (see [Structured Query Expression](#)).

## Request

```
http://<host>/evaluate?expr=<expr>&attributes=<attrs>[&<options>]
```

NAME	VALUE	DESCRIPTION
expr	Text string	Structured query expression that selects a subset of index entities.
attributes	Text string	Comma-delimited list of attributes to include in response.
count	Number (default=10)	Maximum number of results to return.
offset	Number (default=0)	Index of the first result to return.
orderby	Text string	Name of attribute used to sort the results, followed by optional sort order (default=asc): "attrname[:(asc desc)]". If not specified, the results are returned by decreasing natural log probability.
timeout	Number (default=1000)	Timeout in milliseconds. Only results computed before the timeout has elapsed are returned.

Using the *count* and *offset* parameters, a large number of results may be obtained incrementally over multiple requests.

## Response (JSON)

JSONPATH	DESCRIPTION
\$.expr	<i>expr</i> parameter from the request.
\$.entities	Array of 0 or more object entities matching the structured query expression.
\$.aborted	True if the request timed out.

Each entity contains a *logprob* value and the values of the requested attributes.

## Example

In the academic publications example, the following request passes a structured query expression (potentially from the output of an *interpret* request) and retrieves a few attributes for the top 2 matching entities:

```
http://<host>/evaluate?expr=Composite(Author.Name=='jaime  
teeven')&attributes=Title,Y,Author.Name,Author.Id&count=2
```

The response contains the top 2 ("count=2") most likely matching entities. For each entity, the title, year, author name, and author ID attributes are returned. Note how the structure of composite attribute values matches the way they are specified in the data file.

```
{  
  "expr": "Composite(Author.Name=='jaime teeven')",  
  "entities":  
  [  
    {  
      "logprob": -6.645,  
      "Ti": "personalizing search via automated analysis of interests and activities",  
      "Y": 2005,  
      "Author": [  
        {  
          "Name": "jaime teeven",  
          "Id": 1968481722  
        },  
        {  
          "Name": "susan t dumais",  
          "Id": 676500258  
        },  
        {  
          "Name": "eric horvitz",  
          "Id": 1470530979  
        }  
      ]  
    },  
    {  
      "logprob": -6.764,  
      "Ti": "the perfect search engine is not enough a study of orienteering behavior in directed search",  
      "Y": 2004,  
      "Author": [  
        {  
          "Name": "jaime teeven",  
          "Id": 1982462162  
        },  
        {  
          "Name": "christine alvarado",  
          "Id": 2163512453  
        },  
        {  
          "Name": "mark s ackerman",  
          "Id": 2055132526  
        },  
        {  
          "Name": "david r karger",  
          "Id": 2012534293  
        }  
      ]  
    }  
  ]  
}
```

# interpret Method

4/12/2017 • 2 min to read • [Edit Online](#)

The *interpret* method takes a natural language query string and returns formatted interpretations of user intent based on the grammar and index data. To provide an interactive search experience, this method may be called as each character is entered by the user with the *complete* parameter set to 1 to enable auto-complete suggestions.

## Request

```
http://<host>/interpret?query=<query>[&<options>]
```

NAME	VALUE	DESCRIPTION
query	Text string	Query entered by user. If complete is set to 1, query will be interpreted as a prefix for generating query auto-completion suggestions.
complete	0 (default) or 1	1 means that auto-completion suggestions are generated based on the grammar and index data.
count	Number (default=10)	Maximum number of interpretations to return.
offset	Number (default=0)	Index of the first interpretation to return. For example, <i>count=2&amp;offset=0</i> returns interpretations 0 and 1. <i>count=2&amp;offset=2</i> returns interpretations 2 and 3.
timeout	Number (default=1000)	Timeout in milliseconds. Only interpretations found before the timeout has elapsed are returned.

Using the *count* and *offset* parameters, a large number of results may be obtained incrementally over multiple requests.

## Response (JSON)

JSONPATH	DESCRIPTION
\$.query	<i>query</i> parameter from the request.
\$.interpretations	Array of 0 or more ways to match the input query against the grammar.
\$.interpretations[*].logprob	Relative log probability of the interpretation (<= 0). Higher values are more likely.

JSONPATH	DESCRIPTION
\$.interpretations[*].parse	XML string that shows how each part of the query was interpreted.
\$.interpretations[*].rules	Array of 1 or more rules defined in the grammar invoked during interpretation.
\$.interpretations[*].rules[*].name	Name of the rule.
\$.interpretations[*].rules[*].output	Semantic output of the rule.
\$.interpretations[*].rules[*].output.type	Data type of the semantic output.
\$.interpretations[*].rules[*].output.value	Value of the semantic output.
\$.aborted	True if the request timed out.

## Parse XML

The parse XML annotates the (completed) query with information about how it matches against the rules in the grammar and attributes in the index. Below is an example from the academic publications domain:

```
<rule name="#GetPapers">
  papers by
  <attr name="academic#Author.Name" canonical="heungyeung shum">harry shum</attr>
  <rule name="#GetPaperYear">
    written in
    <attr name="academic#Year">2000</attr>
  </rule>
</rule>
```

The `<rule>` element delimits the range in the query matching the rule specified by its `name` attribute. It may be nested when the parse involves rule references in the grammar.

The `<attr>` element delimits the range in the query matching the index attribute specified by its `name` attribute. When the match involves a synonym in the input query, the `canonical` attribute will contain the canonical value matching the synonym from the index.

## Example

In the academic publications example, the following request returns up to 2 auto-completion suggestions for the prefix query "papers by jaime":

```
http://<host>/interpret?query=papers by jaime&complete=1&count=2
```

The response contains the top two ("count=2") most likely interpretations that complete the partial query "papers by jaime": "papers by jaime teevan" and "papers by jaime green". The service generated query completions instead of considering only exact matches for the author "jaime" because the request specified "complete=1". Note that the canonical value "j l green" matched via the synonym "jamie green", as indicated in the parse.

```
{
  "query": "papers by jaime",
  "interpretations": [
    {
      "logprob": -5.615,
      "parse": "<rule name=\"#GetPapers\">papers by <attr name=\"academic#Author.Name\">jaime teevan</attr>
</rule>",
      "rules": [
        {
          "name": "#GetPapers",
          "output": {
            "type": "query",
            "value": "Composite(Author.Name=='jaime teevan')"
          }
        }
      ]
    },
    {
      "logprob": -5.849,
      "parse": "<rule name=\"#GetPapers\">papers by <attr name=\"academic#Author.Name\" canonical=\"j l
green\">jaime green</attr></rule>",
      "rules": [
        {
          "name": "#GetPapers",
          "output": {
            "type": "query",
            "value": "Composite(Author.Name=='j l green')"
          }
        }
      ]
    }
  ]
}
```

When the type of semantic output is "query", as in this example, the matching objects can be retrieved by passing `output.value` to the `evaluate` API via the `expr` parameter.

```
http://<host>/evaluate?expr=Composite(AA.AuN=='jaime teevan')
```

# Structured Query Expression

4/12/2017 • 2 min to read • [Edit Online](#)

A structured query expression specifies a set of operations to evaluate against the data index. It consists of attribute query expressions and higher-level functions. Use the [evaluate](#) method to compute the objects matching the expression. The following is an example from the academic publications domain that returns publications authored by Jaime Teevan since the year 2013.

```
And(Composite(Author.Name=='jaime teevan'),Y>=2013)
```

Structured query expressions may be obtained from [interpret](#) requests, where the semantic output of each interpretation is a structured query expression that returns the index objects matching the input natural language query. Alternatively, they may be manually authored using the syntax described in this section.

## Attribute Query Expression

An attribute query expression identifies a set of objects based on matching against a specific attribute. Different matching operations are supported depending on the attribute type and indexed operation specified in the [schema](#):

TYPE	OPERATION	EXAMPLES
String	equals	Title='latent semantic analysis' (canonical + synonyms)
String	equals	Author.Name=='susan t dumais' (canonical only)
String	starts_with	Title='latent s'...
Int32/Int64/Double	equals	Year=2000
Int32/Int64/Double	starts_with	Year='20'... (any decimal value starting with "20")
Int32/Int64/Double	is_between	Year<2000 Year<=2000 Year>2000 Year>=2000 Year=[2010,2012] (includes only left boundary value: 2010, 2011) Year=[2000,2012] (includes both boundary values: 2010, 2011, 2012)
Date	equals	BirthDate='1984-05-14'
Date	is_between	BirthDate<='2008/03/14' PublishDate=['2000-01-01','2009-12-31']
Guid	equals	Id='602DD052-CC47-4B23-A16A-26B52D30C05B'

For example, the expression "Title='latent s'..." matches all academic publications whose title starts with the string "latent s". In order to evaluate this expression, the attribute Title must specify the "starts\_with" operation in the schema used to build the index.

For attributes with associated synonyms, a query expression may specify objects whose canonical value matches a given string using the "==" operator, or objects where any of its canonical/synonym values match using the "=" operator. Both require the "equals" operator to be specified in the attribute definition.

## Functions

There is a built-in set of functions allowing the construction of more sophisticated query expressions from basic attribute queries.

### And Function

```
And(expr1, expr2)
```

Returns the intersection of the two input query expressions.

The following example returns academic publications published in the year 2000 about information retrieval:

```
And(Year=2000, Keyword=='information retrieval')
```

### Or Function

```
Or(expr1, expr2)
```

Returns the union of the two input query expressions.

The following example returns academic publications published in the year 2000 about information retrieval or user modeling:

```
And(Year=2000, Or(Keyword='information retrieval', Keyword='user modeling'))
```

### Composite Function

```
Composite(expr)
```

Returns an expression that encapsulates an inner expression composed of queries against sub-attributes of a common composite attribute. The encapsulation requires the composite attribute of any matching data object to have at least one value that individually satisfies the inner expression. Note that a query expression on sub-attributes of a composite attribute has to be encapsulated using the Composite() function before it can be combined with other query expressions.

For example, the following expression returns academic publications by "harry shum" while he was at "microsoft":

```
Composite(And(Author.Name="harry shum",
               Author.Affiliation="microsoft"))
```

On the other hand, the following expression returns academic publications where one of the authors is "harry shum" and one of the affiliations is "microsoft":

```
And(Composite(Author.Name="harry shum"),
     Composite(Author.Affiliation="microsoft"))
```

# Linguistic Analysis APIs Overview

4/12/2017 • 1 min to read • [Edit Online](#)

Welcome to the Linguistic Analysis APIs. These APIs provide access to natural language processing (NLP) tools that identify the structure of text. The current release provides three types of analysis:

- [Sentence separation and tokenization](#)
- [Part-of-speech tagging](#)
- [Constituency parsing](#)

## Getting Started

First you can familiar with the major types of analysis listed above. Next, look into the APIs for [getting the list of available analyzers](#) and how they are [named](#). Finally, look into the API for [analyzing new input](#).

# Analyzer Names

4/12/2017 • 1 min to read • [Edit Online](#)

We use a somewhat complicated naming structure for analyzers to allow both flexibility on analyzers and precision in understanding what a name means. Analyzer names consist of four parts: an ID, a Kind, a Specification, and an Implementation. The role of each component is defined below.

## ID

First, an analyzer has a unique ID; a GUID. These GUIDs should change relatively rarely, but are the only way to uniquely describe a particular analyzer.

## Kind

Next, each analyzer is a **kind**. This defines in very broad terms the type of analysis returned, and should uniquely define the data structure used to represent that analysis. Currently, there are three distinct kinds:

- [Tokens](#)
- [POS Tags](#)
- [Constituency Tree](#)

## Specification

Within a given kind, however, different experts might disagree on how a particular phenomenon should be analyzed. Unlike programming languages, there's no clear and exact definition of how this should be done.

For instance, imagine we were trying to find the tokens in the English sentence "He didn't go." In particular, consider the string "didn't". One possible interpretation is that this should be split into two tokens: "did" and "not". Then the alternative sentence "He did not go" would have the same set of tokens. Another possibility is to say that it should be split into the tokens "did" and "n't". The latter token would not normally be considered a word, but this approach retains more information about the surface string, which can sometimes be useful. Or perhaps that contraction should be considered a single word.

Regardless which choice is made, that choice should be made consistently. This is precisely the role of a **specification**: to decide what a correct representation should be.

Analyzer outputs can only be fairly compared to data that conforms to the same specification.

## Implementation

Often there are multiple models that attempt to achieve the same results, but with different performance characteristics. One model might be faster though less accurate; another might make a different trade-off.

The **implementation** portion of an analyzer name is used to identify this type of information, so that users can pick the most appropriate analyzer for their needs.

# Constituency Parsing

4/12/2017 • 2 min to read • [Edit Online](#)

The goal of constituency parsing (also known as "phrase structure parsing") is to identify the phrases in the text. This can be useful when extracting information from text. Customers might want to find feature names or key phrases from a big body of text, and to see the modifiers and actions surrounding each such phrase.

## Phrases

To a linguist, a *phrase* is more than just a sequence of words. To be a phrase, a group of words has to come together to play a specific role in the sentence. That group of words can be moved together or replaced as a whole, and the sentence should remain fluent and grammatical.

Consider the sentence

I want to find a new hybrid automobile with Bluetooth.

This sentence contains the noun phrase: "a new hybrid automobile with Bluetooth". How do we know that this is a phrase? We can rewrite the sentence (somewhat poetically) by moving that whole phrase to the front:

A new hybrid automobile with Bluetooth I want to find.

Or we could replace that phrase with a phrase of similar function and meaning, like "a fancy new car":

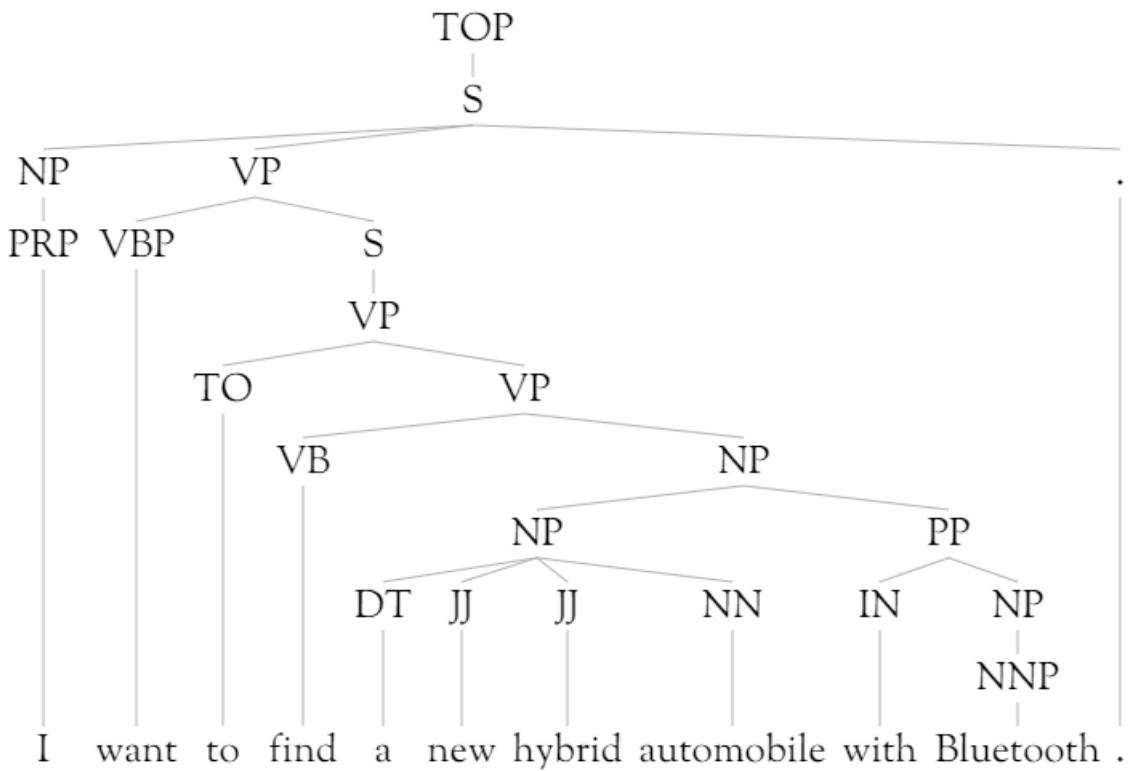
I want to find a fancy new car.

If instead we picked different subset of words, these replacement tasks would lead to strange or unreadable sentences. Consider what happens when we move "find a new" to the front:

Find a new I want to hybrid automobile with Bluetooth.

The results is very difficult to read and understand.

The goal of a parser is to find all such phrases. Interestingly, in natural language, the phrases tend to be nested inside one another. A natural representation of these phrases is a tree, such as the following:



In this tree, the branches marked "NP" are noun phrases. There are several such phrases: *I*, *a new hybrid automobile*, *Bluetooth*, and *a new hybrid automobile with Bluetooth*.

## Phrase Types

LABEL	DESCRIPTION	EXAMPLE
ADJP	Adjective Phrase	"so rude"
ADVP	Adverb Phrase	"clear through"
CONJP	Conjunction Phrase	"as well as"
FRAG	Fragment, used for incomplete or fragmentary inputs	"Highly recommended..."
INTJ	Interjection	"Hooray"
LST	List marker, including punctuation	"#4)"
NAC	Not A Constituent, used to indicate scoping of a non-constituent phrase	"and for a good deal" in "you get things and for a good deal"
NP	Noun Phrase	"a tasty potato pancake"
NX	Used within certain complex NPs to mark the head	
PP	Prepositional Phrase	"in the pool"
PRN	Parenthetical	"(so called)"

LABEL	DESCRIPTION	EXAMPLE
PRT	Particle	"out" in "ripped out"
QP	Quantity Phrase (i.e., complex measure/amount) within a Noun Phrase	"around \$75"
RRC	Reduced Relative Clause.	"still unresolved" in "many issues still unresolved"
S	Sentence or clause.	"This is a sentence."
SBAR	Subordinate clause, often introduced by a subordinating conjunction	"as I left" in "I looked around as I left."
SBARQ	Direct question introduced by a wh-word or -phrase	"What was the point?"
SINV	Inverted declarative sentence	"At no time were they aware." (note how the normal subject "they" was moved to after the verb "were")
SQ	Inverted yes/no question, or main clause of a wh- question	"Did they get the car?"
UCP	Unlike Coordinated Phrase	"small and with bugs" (note how an adjective and a preposition phrase are conjoined with "and")
VP	Verb Phrase	"ran into the woods"
WHADJP	Wh-adjective Phrase	"how uncomfortable"
WHADVP	Wh-adverb Phrase	"when"
WHNP	Wh-noun Phrase	"which potato", "how much soup"
WHPP	Wh-prepositional Phrase	"in which country"
X	Unknown, uncertain, or unbracketable.	first "the" in "the... the soup"

## Specification

Trees here use the S-expressions from the [Penn Treebank](#).

# Part-of-Speech Tagging

4/12/2017 • 2 min to read • [Edit Online](#)

## Background and Motivation

Once a text has been separated into sentences and tokens, the next step of analysis is to identify the category or part-of-speech of each word. These include categories like *noun* (generally representing people, places, things, ideas, etc.) and *verb* (generally representing actions, changes of state, etc.). For some words, the part-of-speech is unambiguous (for instance, *quagmire* is really only a noun), but for many others, it's hard to tell. *Table* could be a place where you sit (or 2-D layout of numbers), but you can also "table a discussion".

## List of Part-of-Speech Tags

TAG	DESCRIPTION	EXAMPLE WORDS
\$	dollar	\$
``	opening quotation mark	`` ``
''	closing quotation mark	'' ''
(	opening parenthesis	( [ {
)	closing parenthesis	) ] }
,	comma	,
--	dash	--
.	sentence terminator	. ! ?
:	colon or ellipsis	: ; ...
CC	conjunction, coordinating	and but or yet
CD	numeral, cardinal	nine 20 1980 '96
DT	determiner	a the an all both neither
EX	existential there	there
FW	foreign word	enfant terrible hoi polloi je ne sais quoi
IN	preposition or subordinating conjunction	in inside if upon whether
JJ	adjective or numeral, ordinal	ninth pretty execrable multimodal
JJR	adjective, comparative	better faster cheaper

TAG	DESCRIPTION	EXAMPLE WORDS
JJS	adjective, superlative	best fastest cheapest
LS	list item marker	(a) (b) 1 2 A B A. B.
MD	modal auxiliary	can may shall will could might should ought
NN	noun, common, singular or mass	potato money shoe
NNP	noun, proper, singular	Kennedy Roosevelt Chicago Weehauken
NNPS	noun, proper, plural	Springfields Bushes
NNS	noun, common, plural	pieces mice fields
PDT	pre-determiner	all both half many quite such sure this
POS	genitive marker	' 's
PRP	pronoun, personal	she he it I we they you
PRP\$	pronoun, possessive	hers his its my our their your
RB	adverb	clinically only
RBR	adverb, comparative	further gloomier grander graver greater grimmer harder harsher healthier heavier higher however larger later leaner lengthier less-perfectly lesser lonelier longer louder lower more ...
RBS	adverb, superlative	best biggest bluntest earliest farthest first furthest hardest heartiest highest largest least less most nearest second tightest worst
RP	particle	on off up out about
SYM	symbol	% &
TO	"to" as preposition or infinitive marker	to
UH	interjection	uh hooray howdy hello
VB	verb, base form	give assign fly
VBD	verb, past tense	gave assigned flew
VBG	verb, present participle or gerund	giving assigning flying
VBN	verb, past participle	given assigned flown

TAG	DESCRIPTION	EXAMPLE WORDS
VBP	verb, present tense, not 3rd person singular	give assign fly
VBZ	verb, present tense, 3rd person singular	gives assigns flies
WDT	WH-determiner	that what which
WP	WH-pronoun	who whom
WP\$	WH-pronoun, possessive	whose
WRB	Wh-adverb	how however whenever where

## Specification

As for tokenization, we rely on the specification from the [Penn Treebank](#).

# Sentence Separation and Tokenization

4/12/2017 • 2 min to read • [Edit Online](#)

## Background and motivation

Given a body of text, the first step of linguistic analysis is to break it into sentences and tokens.

### Sentence Separation

On first glance, it seems that breaking text into sentences is simple: just find the end-of-sentence markers and break sentences there. However, these marks are often complicated and ambiguous.

Consider the following example text:

What did you say?!? I didn't hear about the director's "new proposal." It's important to Mr. and Mrs. Smith.

This text contains three sentences:

- What did you say?!?
- I didn't hear about the director's "new proposal."
- It's important to Mr. and Mrs. Smith.

Note how the ends of sentences are marked in very different ways. The first ends in a combination of question marks and exclamation points (sometimes called an interrobang). The second ends with a period or full stop, but the following quotation mark should be pulled into the prior sentence. In the third sentence, you can see how that same period character can be used to mark abbreviations as well. Looking just at punctuation provides a good candidate set, but further work is required to identify the true sentence boundaries.

### Tokenization

The next task is to break these sentences into tokens. For the most part, English tokens are delimited by white space. (Finding tokens or words is much easier in English than in Chinese, where spaces are mostly not used between words. The first sentence might be written as "Whatdidyousay?")

There are a few difficult cases. First, punctuation often (but not always) should be split away from its surrounding context. Second, English has *contractions*, like "didn't" or "it's", where words have been compressed and abbreviated into smaller pieces. The goal of the tokenizer is to break the character sequence into words.

Let's return to the example sentences from above. Now we've placed a "center dot" (·) between each distinct token.

- What · did · you · say · ?!?
- I · did · n't · hear · about · the · director · 's · " · new · proposal · . · "
- It · 's · important · to · Mr. · and · Mrs. · Smith · .

Note how most tokens are words you'd find in the dictionary (e.g., *important*, *director*). Others solely consist of punctuation. Finally, there are more unusual tokens to represent contractions like *n't* for *not*, possessives like '*s*, etc. This tokenization allows us to handle the word *didn't* and the phrase *did not* in a more consistent way, for instance.

## Specification

It is important to make consistent decisions about what comprises a sentence and a token. We rely on the specification from the [Penn Treebank](#) (some additional details are available here: [\[https://www.cis.upenn.edu/~treebank/tokenization.html\]](https://www.cis.upenn.edu/~treebank/tokenization.html)).

# Analyze Method

4/12/2017 • 2 min to read • [Edit Online](#)

The **analyze** REST API is used to analyze a given natural language input. That might involve just finding the [sentences and tokens](#) within that input, finding the [part-of-speech tags](#), or finding the [constituency tree](#). You can specify which results you want by picking the relevant analyzers. To list all available analyzers, look at the [analyzers](#).

Note that you need to specify the language of the input string.

## REST endpoint:

```
https://westus.api.cognitive.microsoft.com/linguistics/v1.0/analyze
```

## Request parameters

NAME	TYPE	REQUIRED	DESCRIPTION
<b>language</b>	string	Yes	The two letter ISO language code to be used for analysis. For instance, English is "en".
<b>analyzerIds</b>	list of strings	Yes	List of GUIDs of analyzers to apply. See the Analyzers documentation for more information.
<b>text</b>	string	Yes	Raw input to be analyzed. This might be a short string such as a word or phrase, a full sentence, or a full paragraph or discourse.

## Response (JSON)

An array of analysis outputs, one for each attribute specified in the request.

The results look as follows:

NAME	TYPE	DESCRIPTION
analyzerId	string	GUID of the analyzer specified
result	object	analyzer result

Note that the type of the result depends on the input analyzer type.

## Tokens Response (JSON)

NAME	TYPE	DESCRIPTION
result	list of sentence objects	sentence boundaries identified within the text
result[x].Offset	int	starting character offset of each sentence
result[x].Len	int	length in characters of each sentence
result[x].Tokens	list of token objects	token boundaries identified within the sentence
result[x].Tokens[y].Offset	int	starting character offset of the token
result[x].Tokens[y].Len	int	length in characters of the token
result[x].Tokens[y].RawToken	string	the characters inside that token, before normalization
result[x].Tokens[y].NormalizedToken	string	a normalized form of the character, safe for use in a <a href="#">parse tree</a> ; for instance, an open parenthesis character '(' becomes '-LRB-'

Example input: 'This is a test. Hello.' Example JSON response:

```
[
  [
    {
      "Len": 15,
      "Offset": 0,
      "Tokens": [
        {
          "Len": 4,
          "NormalizedToken": "This",
          "Offset": 0,
          "RawToken": "This"
        },
        {
          "Len": 2,
          "NormalizedToken": "is",
          "Offset": 5,
          "RawToken": "is"
        },
        {
          "Len": 1,
          "NormalizedToken": "a",
          "Offset": 8,
          "RawToken": "a"
        },
        {
          "Len": 4,
          "NormalizedToken": "test",
          "Offset": 10,
          "RawToken": "test"
        },
        {
          "Len": 1,
          "NormalizedToken": ".",
          "Offset": 14,
          "RawToken": "."
        }
      ]
    },
    {
      "Len": 6,
      "Offset": 16,
      "Tokens": [
        {
          "Len": 5,
          "NormalizedToken": "Hello",
          "Offset": 16,
          "RawToken": "Hello"
        },
        {
          "Len": 1,
          "NormalizedToken": ".",
          "Offset": 21,
          "RawToken": "."
        }
      ]
    }
  ]
]
```

### POS Tags Response (JSON)

The result is list of lists of strings. For each sentence, there is a list of POS tags, one POS tag for each token. To find the token corresponding to each POS tag, you'll want to ask for a tokenization object as well.

### Constituency Tree Response (JSON)

The result is a list of strings, one parse tree for each sentence found in the input. The parse trees are represented in a parenthesized form.

## Example

```
POST /analyze
```

Request Body: JSON payload

```
{  
  "language": "en",  
  "analyzerIds": [  
    "4FA79AF1-F22C-408D-98BB-B7D7AEEF7F04",  
    "22A6B758-420F-4745-8A3C-46835A67C0D2" ],  
  "text": "Hi, Tom! How are you today?"  
}
```

Response: JSON

```
[  
  {  
    "analyzerId": "4FA79AF1-F22C-408D-98BB-B7D7AEEF7F04",  
    "result": [ [ "NNP", ", ", "NNP", ". "], [ "WRB", "VBP", "PRP", "NN", ". " ] ]  
  },  
  {  
    "analyzerId": "22A6B758-420F-4745-8A3C-46835A67C0D2",  
    "result": [ "(TOP (S (NNP Hi) (, ,) (NNP Tom) (. !)))", "(TOP (SBARQ (WHADVP (WRB How)) (SQ (VP (VBP are))  
    (NP (PRP you)) (NN today) (. ?))))" ]  
  }  
]
```

# Analyzers Method

4/12/2017 • 1 min to read • [Edit Online](#)

The **analyzers** REST API provides a list of analyzers currently supported by the service. The response includes their [names](#) and the languages supported by each (such as "en" for English).

## Request parameters

None

## Response parameters

NAME	TYPE	DESCRIPTION
languages	list of strings	list of two letter ISO language codes for which this analyzer can be used.
id	string	unique ID for this analyzer
kind	string	the broad type of analyzer here
specification	string	the name of the specification used for this analyzer
implementation	string	description of the model and/or algorithm behind this analyzer

## Example

GET /analyzers

Response: JSON

```
[  
  {  
    "id": "22A6B758-420F-4745-8A3C-46835A67C0D2",  
    "languages": ["en"],  
    "kind": "Constituency_Tree",  
    "specification": "PennTreebank3",  
    "implementation": "SplitMerge"  
  },  
  {  
    "id" : "4FA79AF1-F22C-408D-98BB-B7D7AEEF7F04",  
    "languages": ["en"],  
    "kind": "POS_Tags",  
    "specification": "PennTreebank3",  
    "implementation": "cmm"  
  },  
  {  
    "id" : "08EA174B-BFDB-4E64-987E-602F85DA7F72",  
    "languages": ["en"],  
    "kind": "Tokens",  
    "specification": "PennTreebank3",  
    "implementation": "regexes"  
  }  
]
```

# Overview

4/12/2017 • 3 min to read • [Edit Online](#)

## Language Understanding Intelligent Services (LUIS) brings the power of machine learning to your apps

One of the key problems in human-computer interactions is the ability of the computer to understand what a person wants. LUIS is designed to enable developers to build smart applications that can understand human language and accordingly react to user requests. With LUIS, a developer can quickly deploy an HTTP endpoint that will take the sentences sent to it and interpret them in terms of their intents (the intentions they convey) and entities (key information relevant to the intent).

By using LUIS web interface, you can create an application, with a set of intents and entities that are relevant to your application's domain. For example, in a travel agent app, a user might say an utterance like "Book me a ticket to Paris". In this utterance, there is the intention to "BookFlight" and "Paris" is the entity. Intention or the intent can be defined as the desired action and usually contains a verb, in this case "book". The entity is a relevant information of a specific data type, in this case "Paris" is the location entity.

Once your application is deployed and traffic starts to flow into the system, LUIS uses active learning to improve itself. In the active learning process, LUIS identifies the utterances that it is relatively unsure of, and asks you to label them according to intent and entities. This has tremendous advantages; LUIS knows what it is unsure of, and asks for your help in the cases which will lead to the maximum improvement in system performance. LUIS learns quicker, and takes the minimum amount of your time and effort. This is active machine learning at its best.

## Supported Languages

LUIS supports several languages as English, French, Italian, German, Spanish, Brazilian Portuguese, Japanese, Korean and Chinese are supported when it comes to understanding utterances. You choose the culture when you start creating your application, and it cannot be modified once the application is created.

### Chinese support notes

- In the zh-cn culture, LUIS expects the simplified Chinese character set (not the traditional character set).
- The names of intents, entities, features, and regular expressions may be in Chinese or Roman characters.
- When writing regular expressions in Chinese, do not insert whitespace between Chinese characters.

### Rare or foreign words in an application

In the en-us culture, LUIS can learn to distinguish most English words, including slang. In the zh-cn culture, LUIS can learn to distinguish most Chinese characters. If you use a rare word (en-us) or character (zh-cn), and you see that LUIS seems unable to distinguish that word or character, you can add that word or character to a phrase-list feature. For example, Words outside of the culture of the application -- i.e., foreign words -- should be added to a phrase-list feature.

### Tokenization

In normal LUIS use, you won't need to worry about tokenization, but one place where tokenization is important is when manually adding labels to an exported application's JSON file. See the section on importing and exporting an application for details.

To perform machine learning, LUIS breaks an utterance into "tokens". A "token" is the smallest unit that can be labeled in an entity.

How tokenization is done depends on the application's culture:

- **English, French, Italian, Brazilian Portuguese and Spanish:** token breaks are inserted at any whitespace, and around any punctuation.
- **Korean & Chinese:** token breaks are inserted before and after any character, and at any whitespace, and around any punctuation.

## Accessing LUIS programmatically

LUIS offers a set of programmatic REST APIs that can be used by developers to automate the application creation process. These APIs allow you to author and publish your application.

[Click here for a complete API reference.](#)

## Speech Integration

Your LUIS endpoints work seamlessly with [Microsoft Cognitive Service's speech recognition service](#). In the C# SDK for Microsoft Cognitive Services Speech API, you can simply add the LUIS application ID and LUIS subscription key, and the speech recognition result will be sent for interpretation.

See [Microsoft Cognitive Services Speech API Overview](#).

# Plan Your Application

4/12/2017 • 1 min to read • [Edit Online](#)

All LUIS applications are centered around a domain-specific topic, for example booking of tickets, flights, hotels, rental cars etc. or content related to exercising, tracking fitness efforts and setting goals. It is advisable to plan your application before you start creating it in LUIS. Prepare an outline (schema) of the possible intents and entities that are relevant to the domain-specific topic of your application.

Let's take the example of a virtual travel booking agency application. You should think about the intents and entities that are important to your application's task. In a travel booking application, users would like to book a flight and get an idea of the weather at their travel destination. Thus, the relevant intents would be "BookFlight" and "GetWeather". For flight booking, some relevant information is needed such as the location, date, airline, tickets' category and travel class. Thus, they can be added as entities.

With a planned outline of intents and entities, you can start creating your application in LUIS and define these intents and entities.

[Click here](#) to know how to create a new app.

# Create a New App

4/12/2017 • 1 min to read • [Edit Online](#)

You can create and manage your applications on **My Apps** page. You can always access this page by clicking **My Apps** on the top navigation bar of LUIS web page. We are going to create the TravelAgent app and use it as an example application to work on throughout LUIS help topics.

## To create a new app:

1. On **My Apps** page, click **New App**.
2. In the dialog box, name your application "TravelAgent".

### Create a new app

Name (REQUIRED)

Culture (REQUIRED)

▼

\* App culture is the language that your app understands and speaks, not the interface language.

Description (OPTIONAL)

Key to use (OPTIONAL)

▼

**Create**

3. Choose your application culture (for TravelAgent app, we'll choose English), and then click **Create**.

#### NOTE

The culture cannot be changed once the application is created.

Luis creates the TravelAgent app and opens its main page which looks like the following screen. Use the navigation links in the left panel to move through your app pages to define data and work on your app. Your first task in the app is to add intents. For more info on how to add intents, see [Add intents](#).

[Language Understanding](#)[My apps](#)[My keys](#)[Docs](#)[Pricing](#)[Support](#)[About](#)

## TravelAgent

## Dashboard

Facts & statistics about the app's data and the received endpoint hits at any period of time ... [Learn more](#)

### Dashboard

[Intents](#)[Entities](#)[Features](#)[Train & Test](#)[Publish App](#)[← Back to App list](#)

ⓘ You have no intents yet. Intents are the building blocks of your app; they link user requests with the actions that should be taken by your app. Get started by creating your first intent.

[Create an intent](#)[Next tip](#)

App status	Last train: Not trained yet	Last published: Not published yet
Intent Count	0 / 10	Prebuilt Entity Count 0 / 5

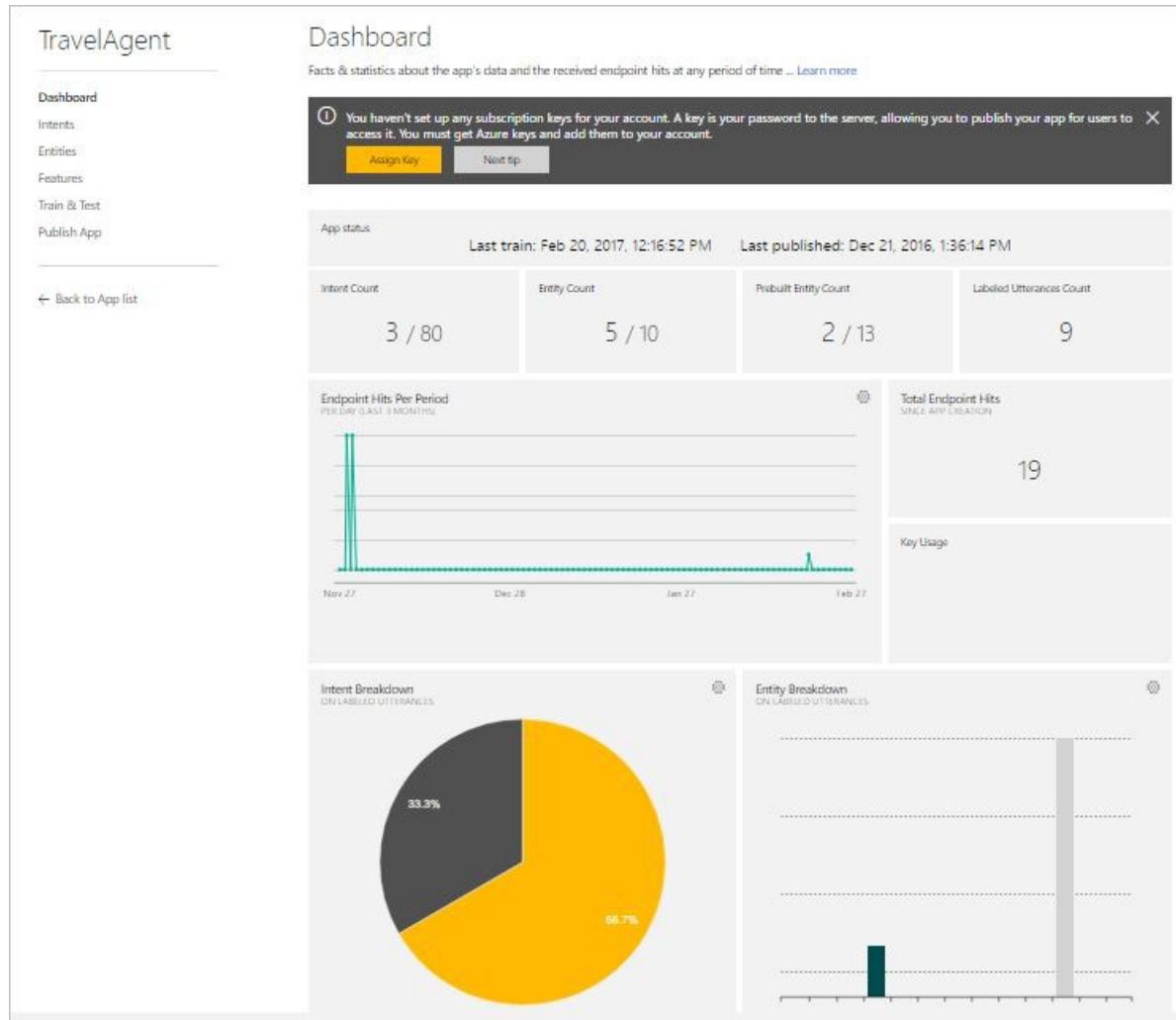
Endpoint Hits Per Period PER DAY (LAST WEEK)	Total Endpoint Hits SINCE APP CREATION
	

# Application Dashboard

4/12/2017 • 3 min to read • [Edit Online](#)

The app dashboard is a visualized reporting tool which enables you to monitor your app at a single glance. The **Dashboard** is the main page that is displayed when you open an app by clicking the application name on **My Apps** page. Also, you can access the **Dashboard** page from inside your app by clicking **Dashboard** in the application's left panel.

The **Dashboard** page gives you an overview of the app and displays significant data compiled from multiple app pages. Some data are analyzed and visualized by graphs and charts to help you get more insight into the app. The dashboard incorporates the latest updates in the app up to the moment and reflects any model changes. The screenshot below shows the **Dashboard** page.



At the top of the **Dashboard** page, a contextual notification bar constantly displays notifications to update you on the required or recommended actions appropriate for the current state of your app. It also provides useful tips and alerts as needed. Below is a detailed description of the data reported on the **Dashboard** page.

## App Status

The dashboard displays the application's training and publishing status, including the date and time when the app was last trained and published.

#### App Status

Last train: Nov 30, 2016, 7:45:49 AM

Last publish: Nov 28, 2016, 8:30:45 AM

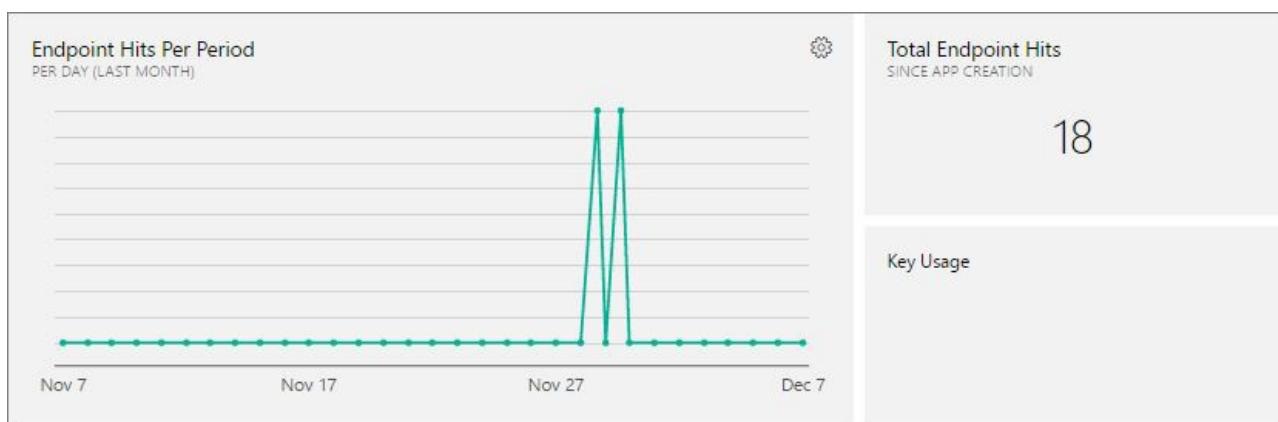
## Model Data Statistics

The dashboard displays the total numbers of intents, entities & labeled utterances existing in the app.

Intent Count	Entity Count	Prebuilt Entity Count	Labeled Utterances Count
3 / 80	5 / 10	2 / 5	9

## Endpoint Hits

The dashboard displays the total endpoint hits received to the app and enables you to display hits within a period that you specify.



### Total endpoint hits

The total number of endpoint hits received to your app since app creation up to the current date.

### Endpoint hits per period

The number of hits received within a past period, displayed per day. A visualized line chart shows the period span from the calculated start date up to the current date (end date). The points between the start and end dates represent the days falling in this period. Hover your mouse pointer over each point to see the hits count in each day within the period. The screenshot below shows the line chart.



To select a period to view its hits on the chart:

1. Click **Additional Settings** to access the periods list. You can select periods ranging from one week up to one year beforehand.

The screenshot shows a dropdown menu titled "Select Period:" with the following options: "Last Month", "Last Week", "Last Month" (which is highlighted in blue), "Last 3 Months", "Last 6 Months", and "Last Year". A back arrow icon is located at the top right of the dashboard.

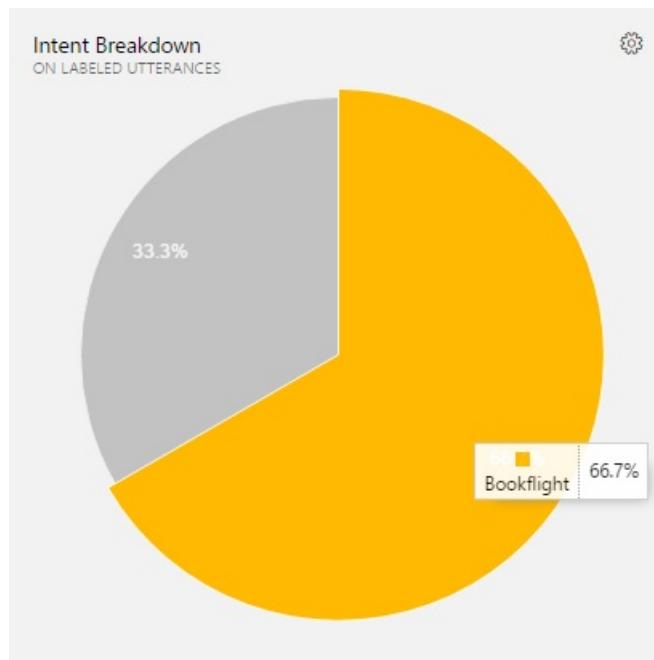
2. Select a period from the list and then click the back arrow to display its hits on the chart.

#### Key usage

The number of hits consumed from the application's subscription key. For more details about subscription keys, see [Manage your keys](#).

## Intent Breakdown

The dashboard displays a breakdown of intents based on labeled utterances or endpoint hits. It visualizes the distribution of intents across labeled utterances/endpoint hits, so that you can see the relative importance of each intent in the app. The intent breakdown is visualized by a pie chart with the slices representing intents. When you hover your mouse pointer over a slice, you'll see the intent name and the percentage it represents of the total count of labeled utterances/endpoint hits.



To control whether the breakdown is based on labeled utterances or endpoint hits:

1. Click **Additional Settings** to access the list as in the screenshot below.

←

## Intent Breakdown

---

Breakdown based on:

Labeled utterances

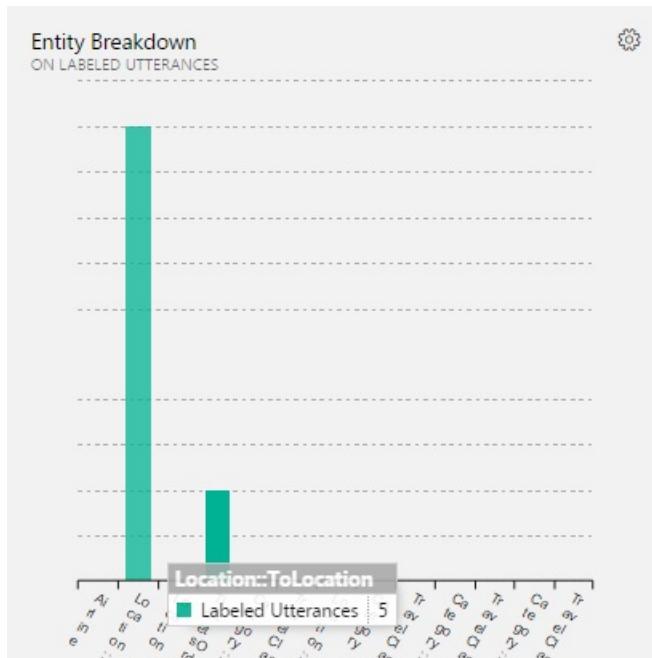
Endpoint hits

Labeled utterances

2. Select a value from the list and then click the back arrow  to display the chart accordingly.

## Entity Breakdown

The dashboard displays a breakdown of entities based on labeled utterances or endpoint hits. It visualizes the distribution of entities across labeled utterances/endpoint hits, showing the usage of each entity in labeled utterances/endpoint hits compared to the other entities. The entity breakdown is visualized by a column chart where entities are displayed along the horizontal axis while their count in labeled utterances/endpoint hits along the vertical axis. When you hover your mouse pointer over a rectangular bar, you'll see the entity name and its count (number of occurrences) in labeled utterances/endpoint hits.



To control whether the breakdown is based on labeled utterances or endpoint hits:

1. Click **Additional Settings**  to access the list as in the screenshot below.



## Entity Breakdown

Breakdown based on:

Labeled utterances

Endpoint hits

Labeled utterances

2. Select a value from the list and then click the back arrow to display the chart accordingly.

# Add Intents

4/12/2017 • 1 min to read • [Edit Online](#)

Intents are the intentions or desired actions conveyed through the utterances (sentences). Intents match user requests with the actions that should be taken by your app. So, you must add intents to help your app understand user requests and react to them properly.

All applications come with the predefined intent, "**None**". You should teach it to recognize user statements that are irrelevant to the app, for example if a user says "Get me a great cookie recipe" in a TravelAgent app.

You can add up to 80 intents in a single LUIS app. You add and manage your intents from the **Intents** page that is accessed by clicking **Intents** in your application's left panel.

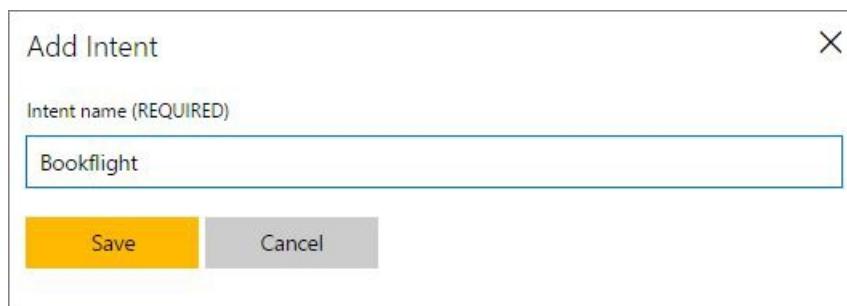
In the following procedure, we'll start adding the "Bookflight" intent in the TravelAgent app as an example to explain the steps of adding an intent.

## To add intent:

1. Open your app (e.g. TravelAgent) by clicking its name on **My Apps** page, and then click **Intents** in the left panel.
2. On the **Intents** page, click **Add intent**.

The screenshot shows the Microsoft Cognitive Services portal interface. At the top, there's a navigation bar with links for Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. On the far right, there's a sign-in link for Carol Hanna and a 'Sign out' button. Below the navigation bar, the Microsoft Cognitive Services logo is displayed. The main content area is titled 'TravelAgent' and 'Intents'. It contains a message: 'A listing of intents in the application. Click an intent to view/edit its details, or add a new intent ... [Learn more](#)'. Below this, there's a yellow 'Add Intent' button. A search bar with placeholder text 'Search for intent ...' is also present. The main table lists one intent: 'None' under 'Intent Name' and '0' under 'Utterances'. At the bottom, a message reads 'No custom intents yet. Add your first intent now.' A sidebar on the left lists other features: Dashboard, Intents (which is selected and highlighted in blue), Entities, Features, Train & Test, and Publish App. At the very bottom of the sidebar is a link to 'Back to App list'.

3. In the **Add Intent** dialog box, type the intent name "BookFlight" and click **Save**.



This will take you directly to the intent details page of the newly added intent "Bookflight", like the screenshot below, in order to add utterances for this intent. For instructions on adding utterances, see [Add example utterances](#).

Language Understanding

My apps

My keys

Docs

Pricing

Support

About



## TravelAgent

### Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Dashboard

#### Intents

Entities

Features

Train & Test

Publish App

Utterances Entities in use Suggested utterances

Type a new utterance & press Enter ...	X
<input type="button" value="Save"/> <input type="button" value="Discard"/> <input type="button" value="Delete"/> <input type="button" value="Reassign Intent"/> <span style="float: right;">Labels view (Ctrl+E): Entities <input type="button" value="▼"/> Search in utterances ... <input type="button" value="○"/></span>	
□ Utterance text	Predicted Intent

[← Back to App list](#)

## Manage your intents

You can view a list of all your intents and manage them on the **Intents** page, where you can add new intents, rename and delete existing ones or access intent details for editing.

[My apps](#) > [TravelAgent](#) > Intents

### Intents

A listing of intents in the application. Click an intent to view/edit its details, or add a new intent ... [Learn more](#)

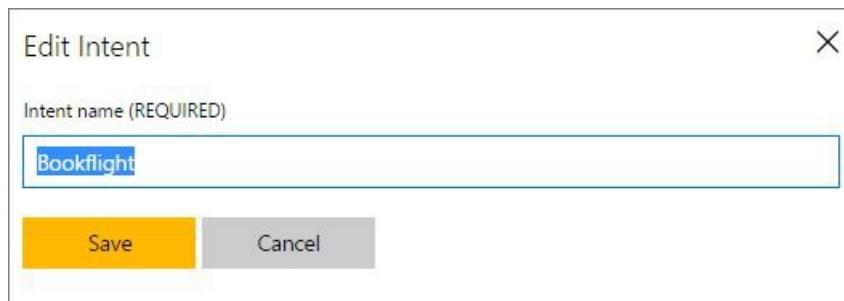
[Add Intent](#)

Search for intent ... X

Intent Name	Utterances	
Bookflight	0	
GetWeather	0	
None	0	

#### To rename an intent:

- On the **Intents** page, click the Rename icon next to the intent you want to rename.
- In the **Edit Intent** dialog box, edit the intent name and click **Save**.



#### To delete an intent:

- On the **Intents** page, click the trash bin icon next to the intent you want to delete.

#### To access intent details for editing:

- On the **Intents** page, click the intent name which you want to access its details.

After adding intents to your app, now your next task is to start adding example utterances for the intents you've added. For instructions, see [Add example utterances](#).

# Add example utterances

4/12/2017 • 6 min to read • [Edit Online](#)

Utterances are sentences representing examples of user queries or commands that your application is expected to receive and interpret. You need to add example utterances for each intent in your app. LUIS learns from these utterances and your app is able to generalise and understand similar contexts. By constantly adding more utterances and labeling them, you are enhancing your application's language learning experience.

For each intent, add example utterances that trigger this intent and include as many utterance variations as you expect users to say. The more relevant & diverse examples you add to the intent, the better intent prediction you'll get from your app. For example, the utterance "book me a flight to Paris" may have variations such as "Reserve me a flight to Paris", "book me a ticket to Paris", "Get me a ticket to Paris", "Fly me to Paris", and "Take me on a flight to Paris".

Utterances are added to an intent on the **Utterances** tab of the intent page. The following steps describe how to add example utterances to an intent (e.g. "BookFlight" intent in the TravelAgent app).

## To add utterance:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Intents** in the left panel.
2. On the **Intents** page, click the intent name "BookFlight" to open its details page, with **Utterances** as the current tab, like the screen below.

The screenshot shows the Microsoft Cognitive Services Language Understanding interface. At the top, there's a navigation bar with links for Microsoft Cognitive Services, Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. On the right, there's a sign-in link for Carol Hanna and a 'Sign out' button. Below the navigation bar, the main area has a sidebar on the left with links for Dashboard, Intents (which is selected), Entities, Features, Train & Test, and Publish App. The main content area is titled 'TravelAgent' and 'Bookflight'. It says, 'Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)'. Below this, there are tabs for Utterances, Entities in use, and Suggested utterances. A text input field says 'Type a new utterance & press Enter ...'. Below it, there are buttons for Save, Discard, Delete, and Reassign Intent. A dropdown menu for 'Labels view (Ctrl+E)' is set to 'Entities'. There's also a search bar for 'Search in utterances ...'. At the bottom, there's a section for 'Predicted Intent' with a checkbox labeled 'Utterance text'. At the very bottom of the sidebar, there's a link to 'Back to App list'.

3. Type "book me 2 adult business tickets to Paris tomorrow on Air France" as a new utterance in the text box, and then press Enter. Note that LUIS converts all utterances to lower case.

The screenshot shows the Microsoft Cognitive Services Language Understanding interface. At the top, there are links for Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. On the right, there is a sign-out link for Carol Hanna. The main area is titled 'TravelAgent' and 'Bookflight'. It says, 'Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... Learn more'. Below this are tabs for Utterances, Entities in use, and Suggested utterances. A search bar contains the utterance 'book me 2 adult business tickets to Paris tomorrow on Air France'. There are buttons for Save, Discard, Delete, and Reassign Intent. A dropdown menu for Labels view (Ctrl+E) is set to Entities. A search bar for 'Search in utterances' is also present. The bottom section shows a table with one row: 'Utterance text' (checkbox checked) and 'Predicted Intent' (N.A Bookflight).

4. Repeat the previous step to add more example utterances.

5. Click **Save** to save the added utterances in the utterances list.

Utterances are added to the utterances list in the current intent. To delete one or more utterances from the list, select them and click **Delete**.

## Label utterances

After adding utterances, your next step is to label them. Utterances are labeled in terms of intents and entities.

### Intent label

Adding an utterance in an intent page means that it is labeled under this intent. That's how an utterance gets an intent label. You can change the intent label of one or more utterances by moving them to another intent. To do this, select the utterances, click **Reassign Intent**, and then select the intent where you want to move them.

### Entity label

There are different types of entities; custom entities and prebuilt entities. You need to label custom entities only, **because prebuilt entities are detected and labeled automatically by your app**.

For example, in the utterance "book me 2 adult business tickets to Paris tomorrow on Air France" that you've just added to "Bookflight" intent in TravelAgent app, before you start labeling entities in this utterance, if you have already added number and datetime as prebuilt entities, you'll notice that "2" and "tomorrow" were automatically detected as prebuilt entities, where "2" is labeled as "number" and "tomorrow" as "datetime". This will look like the following screenshot.

The screenshot shows the Microsoft Cognitive Services Language Understanding interface. At the top, there are links for My Apps, TravelAgent, Intents, and a Save/Discard button. The main area is titled 'Bookflight'. It says, 'Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... Learn more'. Below this are tabs for Utterances, Entities in use, and Suggested utterances. A search bar contains the utterance 'book me [number] adult business tickets to paris [\$datetime] on air france'. There are buttons for Save, Discard, Delete, and Reassign Intent. A dropdown menu for Labels view (Ctrl+E) is set to Entities. A search bar for 'Search in utterances' is also present. The bottom section shows a table with one row: 'Utterance text' (checkbox checked) and 'Predicted Intent' (N.A Bookflight).

To Learn more about prebuilt entities and how to add them, see [Add entities](#).

In the following procedure, we'll label custom entities (simple, hierarchical and composite entities) in the utterance "book me 2 adult business tickets to Paris tomorrow on Air France".

1. Select "Air France" in the utterance mentioned above to label it as entity.

#### NOTE

When selecting words to label them as entities:

- For a single word, just click it.
- For a set of two or more words, click at the beginning and then at the end of the set.

2. In the entity drop-down box that appears, you can either click an existing entity (if available) to select it, or add a new entity by typing its name in the text box and clicking **Create entity**. Now, we'll create the simple entity "Airline". Type "Airline" in the text box and then click **Create entity**.

[My Apps](#) > [TravelAgent](#) > [Intents](#) > Bookflight

## Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

[Utterances](#) [Entities in use](#) [Suggested utterances](#)

Type a new utterance & press Enter ... X

[Save](#) [Discard](#) [Delete](#) [Reassign Intent](#) ▾ Labels view (Ctrl+E): [Entities](#) ▾ [Search in utterances ...](#)

□	Utterance text	Predicted Intent
<input checked="" type="checkbox"/>	book me [\$number] adult business tickets to paris tomorrow on [air france]	N.A Bookflight

1

🔍

Airline

+ Create entity

#### NOTE

This way is used to create a simple entity on the spot (while labeling utterances). More complicated entities (e.g. hierarchical and composite) can only be created from the **Entities** page. For more instructions, see [Add entities](#).

3. Click "Paris" in the same utterance, then click "ToLocation" in the entity drop-down box as the entity label. "ToLocation" is a hierarchical entity that must be added on the **Entities** page. To learn more about hierarchical entities and how to add them, see [Add entities](#).

## BookFlight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

[Utterances](#) [Entities in use](#) [Suggested utterances](#)

Manage utterances of this intent, in a customizable list view, and take appropriate actions to improve your model ... [Learn more](#)

The screenshot shows the 'BookFlight' intent management interface. An utterance is selected: "book me [\$number] adult business tickets to [paris] [\$datetime] on [\$Airline]". The 'Predicted Intent' is 'N.A' under 'BookFlight'. A dropdown menu is open over the entity placeholder '\$number', showing options like 'Search/Add entity ...', 'Back', 'Location::FromLocation', and 'Location::ToLocation'.

4. Click "2" (labeled as "number") and then click **Remove label** in the drop-down box. We remove this label as we do not want "2" to be interpreted individually, but to be part in a composite entity that we're going to label now.
5. Select the phrase "2 adult business" by clicking at the beginning and at the end of the phrase, then click "TicketsOrder" in the drop-down box. "TicketsOrder" is a composite entity that must be added on the **Entities** page. To learn more about composite entities and how to add them, see [Add entities](#).

## BookFlight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

[Utterances](#) [Entities in use](#) [Suggested utterances](#)

Manage utterances of this intent, in a customizable list view, and take appropriate actions to improve your model ... [Learn more](#)

The screenshot shows the 'BookFlight' intent management interface. An utterance is selected: "book me [2 adult business] tickets to [Location::ToLocation] [\$datetime] on [\$Airline]". The 'Predicted Intent' is 'N.A' under 'BookFlight'. A dropdown menu is open over the entity placeholder '[2 adult business]', showing options like 'Search/Add entity ...', 'Airline', and 'TicketsOrder (Composite)'. Below the dropdown, there are two sections: 'Location (Hierarchical)' and 'TravelClass (Hierarchical)' each with a right-pointing arrow.

6. Click **Save**.

### To remove an entity label:

- Click the entity you want to remove its label and click **Remove label** in the entity drop-down box that appears. Then, click **Save** to save this change.

## Search in utterances

Searching allows you to find utterances that contain a specific text (words/phrases). For example, sometimes you will notice an error that involves a particular word, and may want to find all the examples including it.

## To search in utterances:

- Type the search text in the search box at the top right corner of the utterances list and press Enter. The utterances list will be updated to display only the utterances including your search text. For example, in the following screenshot, only the utterances which contain the search word "reserve" is displayed.

The screenshot shows a table of utterances. At the top, there are buttons for Save, Discard, Delete, and Reassign Intent. To the right of these is a search bar labeled 'Labels view (Ctrl+E)' with dropdown menus for 'Entities' and 'reserve'. Below the search bar is a magnifying glass icon. The table has columns for 'Utterance text', 'Predicted Intent', and a numerical confidence score. There are three rows of data:

Utterance text	Predicted Intent	
reserve a ticket to [\${Location::ToLocation}]	0.99 Bookflight	
reserve {2 adult economy} tickets to london	N.A Bookflight	

A green box highlights the number '1' at the bottom center of the table.

To cancel the search and restore your full list of utterances, delete the search text you've just typed.

## Filter utterances

When you have a large number of utterances, it is useful to filter utterances in order to limit their view based on one or more filtering criteria.

You can apply one or more filters on utterances, as needed. These are the available filters that you can use:

- Selected:** displays only the currently selected utterances.
- Changed:** displays only utterances which contain unsaved changes.
- Errors:** displays only utterances which contain errors.
- Entity:** displays only utterances that contain a specific entity.

### NOTE

Utterances which contain unsaved changes are highlighted in light yellow. You can click **Save** to save changes or **Discard** to discard changes.

## To apply filter(s):

- Click the filter button at the top right corner of the utterances list, to display all filters.
- Click on the filter(s) that you want to apply on utterances. For the **Entity** filter, select the entity by which you want to filter utterances.

## Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (6) Entities in use (2) Suggested utterances

The screenshot shows the 'Bookflight' intent management interface. At the top, there are buttons for Save, Discard, Delete, and Reassign Intent. To the right, there is a 'Labels view (Ctrl+E)' dropdown set to 'Entities', a search bar, and a filter icon. Below this, there are two green buttons: 'Selected' and 'Changed'. The main area displays two utterances:

- book me {[\$number] adult business} tickets to paris [\$datetime] on air france** (Predicted Intent: BookFlight, Category: Bookflight, Location: Airline, TicketsOrder: 0.9)
- reserve a ticket to [\$Location::ToLocation]** (Predicted Intent: BookFlight, TravelClass: 0.9)

On the right side, there is a sidebar with the following labels:

- Predicted Intent: Airline
- Category: Category
- Location: Location
- TicketsOrder: TicketsOrder
- TravelClass: TravelClass
- datetime: datetime
- number: number

The applied filters appear as green buttons at the top left corner of the utterances list.

- To clear an applied filter, click its button at the top left corner.
- To clear all applied filters, click all their corresponding buttons, or just click the filter button .

## Choose labels view in utterances

You can control how you see the words labeled as entities in the utterances by selecting one of the available views for labeled entities. At the top of the utterances list, select a view from the **Labels view** list. These are the available views:

- Entities:** Shows entity-labeled words in tagged format (entity labels), enclosed in square brackets, with only composite entities displayed as normal text between curly brackets.

The screenshot shows the 'Bookflight' intent management interface with the 'Entities' label view selected in the dropdown. The utterances are displayed as follows:

- book me {2 adult business} tickets to [\$Location::ToLocation] [\$datetime] on [\$Airline]** (Predicted Intent: BookFlight, N.A.: N.A.)

- Tokens:** Shows all entity-labeled words in text format (normal text), enclosed in square brackets except composite entities in curly brackets.

The screenshot shows the 'Bookflight' intent management interface with the 'Tokens' label view selected in the dropdown. The utterances are displayed as follows:

- book me {2 adult business} tickets to [paris] [tomorrow] on [air france]** (Predicted Intent: BookFlight, N.A.: N.A.)

- Composite entities:** Shows only the words labeled as composite entities in tagged format (entity labels), enclosed in curly brackets.

Save Discard Delete Reassign Intent

Labels view (Ctrl+E): Composite entities ▾ Search in utterances ...

Utterance text

book me {\$TicketsOrder } tickets to paris tomorrow on air france

Predicted Intent: N.A BookFlight

1

This screenshot shows a user interface for managing utterances in a conversational AI system. At the top, there are buttons for Save, Discard, Delete, and Reassign Intent. To the right, there's a dropdown menu for 'Labels view (Ctrl+E)' with options: Composite entities (which is selected and highlighted in blue), Tokens, Entities, and another Composite entities option. There's also a search bar labeled 'Search in utterances ...' with a magnifying glass icon. Below this, there's a section for 'Utterance text' containing the input 'book me {\$TicketsOrder } tickets to paris tomorrow on air france'. To the right of the text, it says 'Predicted Intent' followed by 'N.A' and 'BookFlight'. A small green box at the bottom left contains the number '1', likely indicating the count of utterances or a specific identifier.

You can press **Ctrl+E** to quickly switch between views.

# Add Entities

4/12/2017 • 6 min to read • [Edit Online](#)

Entities are key data in your application's domain. An entity represents a class including a collection of similar objects (places, things, people, events or concepts). Entities describe information relevant to the intent, and sometimes they are essential for your app to perform its task. For example, a News Search app may include entities such as "topic", "source", "keyword" and "publishing date", which are key data to search for news. In a travel booking app, the "location", "date", "airline", "travel class" and "tickets" are key information for flight booking (relevant to the "Bookflight" intent). So, we'll add them as entities.

You do not need to create entities for every concept in your app, but only for those required for the app to take action. You can add up to **10** entities in a single LUIS app.

You can add, edit or delete entities in your app through the **Entities list** on the **Entities** page. Luis offers many types of entities; prebuilt entities, custom machine learned entities and close list entities.

## Add Prebuilt Entities

LUIS provides a set of prebuilt (system-defined) entities, covering many examples of the most common knowledge concepts such as date, age, temperature, percentage, dimension, cardinal and ordinal numbers, etc.

For example, the TravelAgent app may receive a request like "Book me a flight to Boston on May 4". This will require your app to understand date and time words in order to interpret the request properly. Rather than creating entities for such concepts from scratch, you can enable a ready-made prebuilt entity called "datetime".

As another example, a travel booking request like "Book me the first flight to Boston" might be sent to your app. This requires understanding of ordinal words (e.g. first, second). Therefore, you must add a prebuilt entity called "ordinal", for your app to recognize ordinal numbers.

For a full list of prebuilt entities and their use, see [Prebuilt Entities List](#).

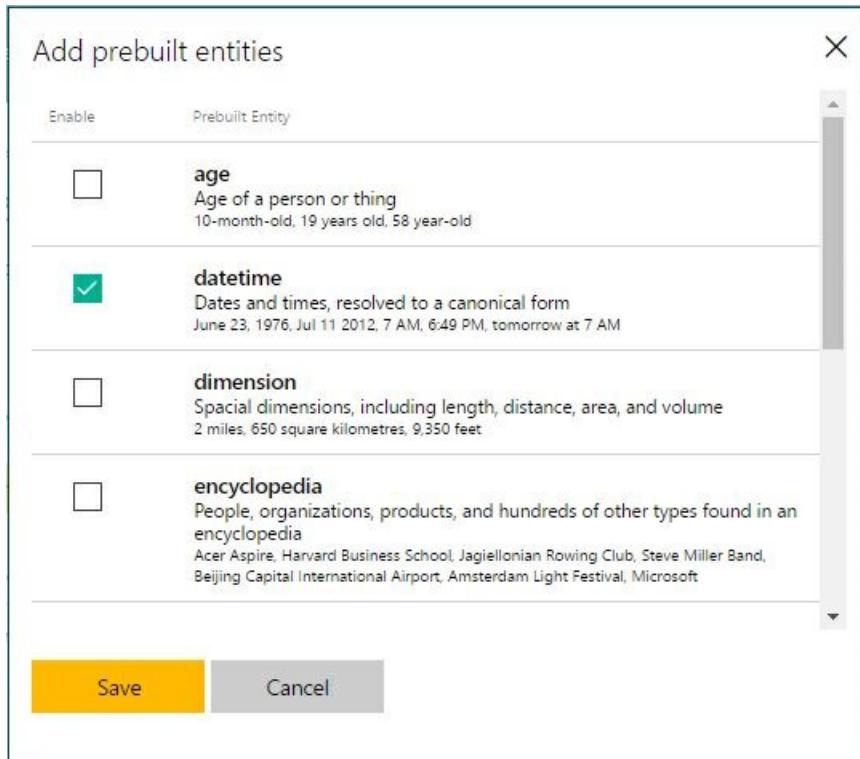
### To add a prebuilt entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Entities** in the left panel.
2. On the **Entities** page, click **Add prebuilt entity**.

The screenshot shows the Microsoft Cognitive Services portal. At the top, there's a navigation bar with the Microsoft logo, 'Cognitive Services', and user info ('Thanaa Al Shamy' and 'Sign out'). Below the navigation is a dark green header with links for 'Language Understanding', 'My Apps', 'My Keys', 'Docs', 'Pricing', 'Support', and 'About'. To the right of the header are icons for 'Profile' and 'Help'. The main content area has a breadcrumb trail: 'My apps > TravelAgent > Entities'. The title 'Entities' is displayed above a sub-header: 'Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)'. Below this are three buttons: 'Entities list', 'Labeled utterances', and 'Suggested utterances'. A note says: 'Here's a list of all entities in your app. Add and refine entities for a more precise capture of key data ... [Learn more](#)'. At the bottom of the list area are two buttons: 'Add custom entity' (highlighted in yellow) and 'Add prebuilt entity'. A search bar at the bottom has 'Entity Name' and 'Entity Type' fields. A message at the bottom states: 'No entities yet. Add your first new entity now.'

3. In **Add prebuilt entities** dialog box, click the prebuilt entity you want to add (e.g. "datetime"), and then

click **Save**.



## Add Custom Entities

Custom entities are the entities you create in your app. There are three types of custom entities:

- **Simple:** a generic entity.
- **Hierarchical:** a parent including children (sub-types) which are dependent on the parent.
- **Composite:** a compound of two or more separate entities combined together forming a composite and treated as a single entity.

### Simple Entities

A simple entity is a generic entity that describes a single concept. In the example of the TravelAgent app, a user may say "Book me a flight to London tomorrow on British Airways", where "British Airways" is the name of an airline company. In order to capture the notion of airline names, let's create the entity "Airline".

#### To add a simple entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Entities** in the left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "Airline" in the **Entity name** box, select **Simple** from the **Entity type** list, and then click **Save**.

Add Entity X

Entity name (REQUIRED)

Entity type (REQUIRED)

**Save**

**Cancel**

## Hierarchical Entities

You can define relationships between entities based on hereditary hierarchical patterns, where the generic entity acts as the parent and the children are sub-types under the parent, and they share the same characteristics. For example, in the TravelAgent app, you can add three hierarchical entities:

- "Location", including the entity children "FromLocation" and "ToLocation", representing source and destination locations.
- "TravelClass", including the travel classes as children ("first", "business" and "economy")
- "Category", including ticket categories ("adult", "child" and "infant").

Do the following steps to add hierarchical entities and make sure to add the children at the same time you are creating the parent entity. You can add up to 10 entity children for each parent.

### To add a hierarchical entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Entities** in the left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "Location" in the **Entity name** box, and then select **Hierarchical** from the **Entity type** list.

Add Entity X

Entity name (REQUIRED)

Entity type (REQUIRED)

Child # 1



Child # 2



+ Add child

**Save**

**Cancel**

4. Click **Add Child**, and then type "FromLocation" in **Child #1** box.

5. Click **Add Child**, and then type "ToLocation" in **Child #2** box.

**NOTE**

To delete a child (in case of a mistake), click the trash bin icon next to it.

6. Click **Save**.

## Composite Entities

You can also define relationships between entities based on associative patterns by creating "composite entities". A composite entity is created by combining two or more existing entities (simple or hierarchical) and treating them as one entity. Unlike a hierarchical entity, the composite entity and the children forming it are not in a parent-child relationship; they are independent of each other and they do not share common characteristics. The composite pattern enables your app to identify entities, not only individually, but also in groups.

In the TravelAgent app example, a user may say "Book 2 adult business tickets to Paris next Monday". In this example, we can create a composite entity called "TicketsOrder", including three children entities: "number", "category" and "class" which describe the tickets to be booked. Before creating a composite entity, you must first add the entities forming it, if they do not already exist.

### To add the entities forming the composite:

1. Add the prebuilt entity "number". For instructions, see the Add Prebuilt Entities section above.
2. Add the hierarchical entity "Category", including the sub-types: "adult", "child" and "infant", and "TravelClass" including "first", "business" and "economy". For more info, see the Hierarchical Entities section above.

### To add the composite entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page and click **Entities** in the app's left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "TicketsOrder" in the **Entity name** box, and then select **Composite** from the **Entity type** list. Click on the add child link to add new child.
4. In **Child #1**, select the entity "number" from the list.
5. In **Child #2**, select the parent entity "Category" from the list.
6. In **Child #3**, select the parent entity "TravelClass" from the list.

Add Entity

Entity name (REQUIRED)

Entity type (REQUIRED)

Child # 1

Child # 2

Child # 3

+ Add child

**Save** **Cancel**

This screenshot shows the 'Add Entity' dialog box. At the top, there's a title 'Add Entity' and a close button 'X'. Below that, there are two required fields: 'Entity name (REQUIRED)' containing 'TicketsOrder' and 'Entity type (REQUIRED)' set to 'Composite'. Under 'Entity type', there are three child entities listed: 'Child # 1' with value 'number', 'Child # 2' with value 'Category', and 'Child # 3' with value 'TravelClass'. Each child entry has a small trash icon to its right. At the bottom of the dialog are 'Save' and 'Cancel' buttons.

7. Click **Save**.

**NOTE**

To delete a child (in case of a mistake), click the trash icon next to it.

## Edit/Delete Entities

You can edit or delete entities from the **Entities list** on the **Entities** page of your app.

### To edit an entity:

1. On the **Entities** page, click the entity in the **Entities list**.
2. In the **Edit Entity** dialog box, you can edit the entity name and children names, or add more children (for hierarchical/composite entities), but the entity type is not editable.

Edit Entity X

Entity name (REQUIRED)

Entity type (REQUIRED)

Child # 1  
 Delete

Child # 2  
 Delete

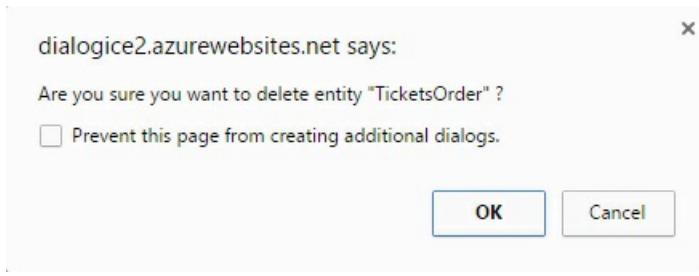
[+ Add child](#)

Save Cancel

3. Click **Save**.

#### To delete an entity:

- In the **Entities list**, click the trash bin icon next to the entity you want to delete. Then, click **OK** in the confirmation message to confirm deletion.



#### NOTE

- Deleting a hierarchical entity deletes all its children entities.
- Deleting a composite entity deletes only the composite and breaks the composite relationship, but doesn't delete the entities forming it.

## Review Labeled Utterances for Entities

To review the labeled utterances that contain a specific entity, click the **Labeled Utterances** tab on the **Entities** page, and choose the entity for which you want to display all labeled utterances. You can modify entity labels in labeled utterances, if required, and then click **Save**.

## Entities

Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)

Entities list Labeled utterances Suggested utterances

Labeled utterances for :

Choose Entity ... ▾

Choose Entity ...

Airline

Category

Location

TicketsOrder

TravelClass

Save Discard

Labels view (Ctrl+E): Entities Search in utterances ...

Predicted Intent



Now that you have added intents, utterances and entities, you have a basic LUIS app ready to be trained and tested for publishing. For more information on how to train and test your app, [click here](#).

# Add Features

4/12/2017 • 3 min to read • [Edit Online](#)

Features help you improve the detection or prediction of intents and entities in utterances. For example, when your app fails to identify an entity, adding a “phrase list” feature with some or all of the entity’s potential values will improve the detection of this entity. Also, adding a “pattern” feature helps your application easily recognize regular patterns that are frequently used in your application’s domain, such as the pattern of flight numbers in a travel app or product codes in a shopping app.

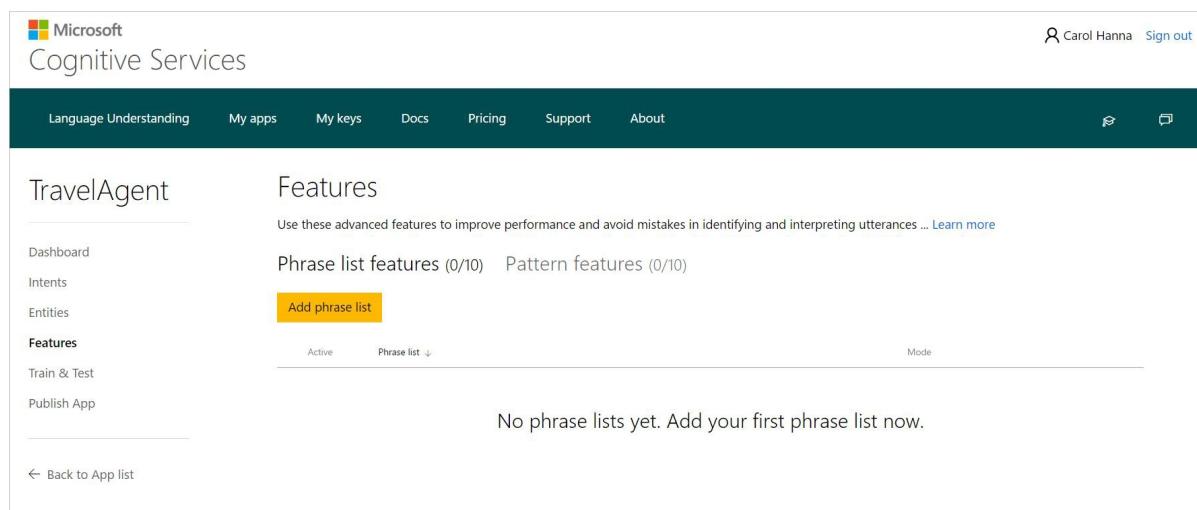
## Phrase list features

You can create a “phrase list” including a group of values (words or phrases) that belong to the same class and must be treated similarly (e.g. names of cities or products), so that what LUIS learns about one of them will be automatically applied to the others as well. For example in the TravelAgent app, London, Paris, Cairo, etc. can be values of a phrase list named as “Cities”. If you label one of these values as an entity, others will mostly be predicted the same.

LUIS may be unable to recognize rare and proprietary words, as well as foreign words (out of the culture of the app), and therefore they should be added to a phrase list feature.

### To add a phrase list:

1. Open your app by clicking its name on **My Apps** page, and then click **Features** in your app's left panel.
2. On the **Features** page, click **Add phrase list**.



The screenshot shows the Microsoft Cognitive Services Language Understanding interface. At the top, there's a navigation bar with links for Microsoft Cognitive Services, Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. On the right, there are user profile icons for Carol Hanna and a 'Sign out' link. Below the navigation bar, the main area displays the 'TravelAgent' app details. On the left, a sidebar lists options like Dashboard, Intents, Entities, **Features** (which is currently selected and highlighted in blue), Train & Test, and Publish App. The main content area has a heading 'Features' and a sub-section 'Phrase list features (0/10)'. A prominent yellow button labeled 'Add phrase list' is centered. Below it, there are tabs for 'Active' and 'Phrase list'. A message at the bottom says 'No phrase lists yet. Add your first phrase list now.' At the very bottom of the sidebar, there's a link to 'Back to App list'.

3. In the **Add Phrase List** dialog box, type "Cities" as the name of the phrase list in the **Phrase list name** text box.
4. In **Phrase list values**, type the values you want to include in the phrase list, separated by **commas** (e.g. London, Paris, Seattle, Berlin, Dubai, Cairo)

Add Phrase list X

Phrase list name (REQUIRED)

Phrase list values (REQUIRED)

Is exchangeable?

Is active?

Save Cancel

5. Click **Is exchangeable** if the added phrase list values are alternatives that can be used interchangeably.
6. Click **Is active** if you want this phrase list to be active (i.e. applicable and used) in your app.
7. Click **Save**. The phrase list will be added to phrase list features on the **Features** page.

#### To edit a phrase list:

- Click the phrase list name in the list of phrase list features. In the **Edit Phrase List** dialog box that opens, make the required editing changes and then click **Save**.

Edit Phrase list X

Phrase list name (REQUIRED)

Phrase list values (REQUIRED)

Is exchangeable?

Is active?

Save Cancel

#### To delete a phrase list:

- Click the trash bin icon  next to the phrase list name in the list of phrase list features.

## Pattern features

You can create a structured “pattern” to represent a certain class of objects (e.g. flight numbers, product codes, etc.). A pattern is defined in regular expression (Regex). This will help LUIS easily recognize the string of the defined pattern in utterances, and thus classify it correctly. For example, in a travel app, flight numbers might follow a regular pattern of two letters followed by three digits.

#### To add a pattern:

1. Open your app by clicking its name on **My Apps** page, and then click **Features** in your app's left panel.
2. On the **Features** page, click the **Pattern Features** tab, and then click **Add Pattern Feature**.

Microsoft Cognitive Services

Carol Hanna Sign out

Language Understanding My apps My keys Docs Pricing Support About

TravelAgent

Features

Use these advanced features to improve performance and avoid mistakes in identifying and interpreting utterances ... [Learn more](#)

Phrase list features (0/10) Pattern features (0/10)

Add pattern feature

Active Pattern feature ↓

No pattern features yet. Add your first pattern feature now.

← Back to App list

3. In the **Add Pattern** dialog box, type "Flight number" in the **Pattern name** text box.
4. To learn about the supported regex syntax, click **learn about supported regex syntax** to expand the dialog and display it, as in the screen below. To collapse the dialog and hide syntax, click it again.

Add Pattern X

Pattern name (REQUIRED)

Pattern value (REQUIRED)

Is active?

[Learn about supported regex syntax ^](#)

Regular expression allowed syntax:

- Use '+' to represent 1 or more repetitions, '\*' for 0 or more repetitions and '?' for 0 or 1 repetitions.
- Use \w to match a word character, \d to match a digit, and "." to match any character.
- Use | to apply a logical OR between characters/groups.
- Use {a,b} to apply a repetition for a character/group.
- Use ^ and \$ to match beginning and end of utterance.

Save Cancel

5. In the **Pattern value** text box, type [A-Za-z]{2}[0-9]{3} as the value of the flight number pattern.

Add Pattern X

Pattern name (REQUIRED)

Pattern value (REQUIRED)

Is active?

[Learn about supported regex syntax ^](#)

Regular expression allowed syntax:

- Use '+' to represent 1 or more repetitions, '\*' for 0 or more repetitions and '?' for 0 or 1 repetitions.
- Use '\w' to match a word character, '\d' to match a digit, and '.' to match any character.
- Use '| to apply a logical OR between characters/groups.
- Use {a,b} to apply a repetition for a character/group.
- Use '^' and '\$' to match beginning and end of utterance.

Save Cancel

6. Click **Is active** if you want this pattern to be active (i.e. applicable and used) in your app. A feature is active by default.
7. Click **Save**. The pattern will be added to pattern features on the **Features** page.

#### To edit a pattern:

- Click the pattern name in the list of pattern features. In the **Edit Pattern** dialog box that opens, make the required editing changes and then click **Save**.

Edit Pattern X

Pattern name (REQUIRED)

Pattern value (REQUIRED)

Is active?

[Learn about supported regex syntax ^](#)

Save Cancel

#### To delete a pattern:

- Click the trash icon button next to the pattern name in the list of pattern features.

# Train and test your app

4/12/2017 • 8 min to read • [Edit Online](#)

## Train to teach your app

You should continuously work on your application to refine it and improve its language understanding. Whenever you make updates by adding, editing or deleting entities, intents or utterances in your current model, you'll need to train your app before testing and publishing. When you "train" a model, LUIS generalizes from the examples you have labeled, and builds model to recognize the relevant intents and entities in the future, thus improving its classification accuracy.

### To train your current model:

1. Access your app (e.g. TravelAgent) by clicking its name on **My Apps** page.
2. In your app, click **Train & Test** in the left panel.
3. On the **Test App** page, click **Train Application** to train the current model on the latest updates.

The screenshot shows the LUIS Test Application interface for the 'TravelAgent' app. On the left, there's a sidebar with navigation links: Dashboard, Intents, Entities, Features, Train & Test (which is currently selected and highlighted in yellow), and Publish App. Below the sidebar is a link to 'Back to App list'. The main area is titled 'Test your application' with a sub-instruction: 'Use this tool to test the current and published versions of your application, to check if you are progressing on the right track ... [Learn more](#)'. A yellow button labeled 'Train Application' is visible. Below it, a status message says 'Last train: Feb 20, 2017, 12:16:51 PM | Last publish: Dec 21, 2016, 1:36:14 PM'. There are two tabs at the top of the main area: 'Interactive Testing' (selected) and 'Batch Testing'. Under 'Interactive Testing', there's a checkbox for 'Enable published model', a text input field for 'Type a test utterance & press Enter', and a large empty text area for results. At the top right of the main area are buttons for 'Labels view (Ctrl+E)', 'Entities' (with a dropdown arrow), and 'Reset console'.

### NOTE

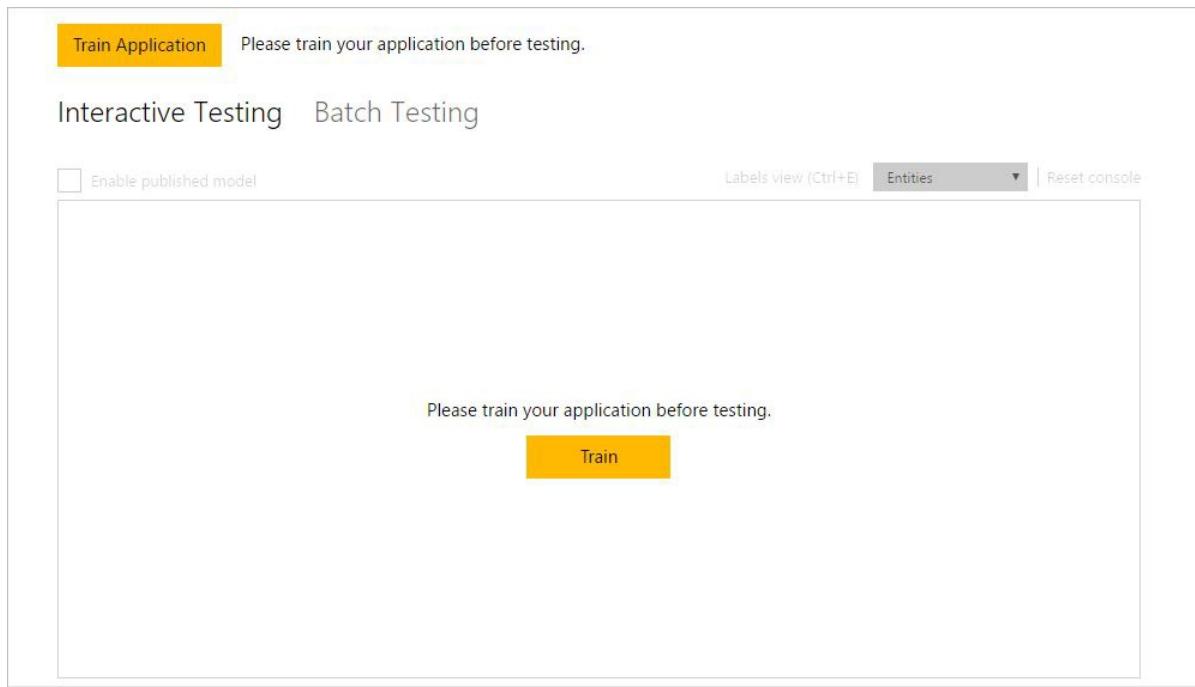
If you have one or more intents in your app which do not contain example utterances, you'll not be able to train your app until you add utterances for all your intents. For more information, see [Add example utterances](#).

## Test your app

Luis provides two types of testing; interactive testing and batch testing. You can use any of them by using the corresponding tab on the **Test App** page.

### To access the Test App page:

1. Access your app (e.g. TravelAgent) by clicking its name on **My Apps** page,
  2. Click **Train & Test** in your application's left panel to access the **Test App** page.
- If you haven't already trained your current model on recent updates, then your test page will look like this screenshot:



- If your model is trained, your test page will look like this screenshot:

A screenshot of the same application interface after the model has been trained. The "Train Application" button is now greyed out. The main area displays the message "Please train your application before testing." above a large yellow "Train" button. On the left, the sidebar shows the "Train &amp; Test" item is selected. The main content area has a title "Test your application" and a sub-section "Interactive Testing". It includes a text input field with placeholder text "Type a test utterance &amp; press Enter" and a "→" button. To the right of the input field is a large, empty rectangular area representing the results panel.

## Interactive Testing

Interactive testing enables you to test both the current and published versions of your app and compare their results in one screen. Interactive testing runs by default on the current trained model only. For a published model, interactive testing is disabled and needs your action to enable it, because it is counted in hits and will be deducted from your key balance.

The **Interactive Testing** tab is divided into two sections (as in the screenshot):

The screenshot shows the LUIS Test App interface. On the left, there's a sidebar with links like Dashboard, Intents, Entities, Features, Train & Test (which is selected), Publish App, and a Back to App list link. The main area has a title 'Test your application' with a sub-instruction: 'Use this tool to test the current and published versions of your application, to check if you are progressing on the right track ... Learn more'. Below this is a yellow button labeled 'Train Application' and text indicating the last train was on Feb 20, 2017, at 12:16:51 PM, and the last publish was on Dec 21, 2016, at 1:36:14 PM. There are tabs for 'Interactive Testing' and 'Batch Testing'. Under 'Interactive Testing', there's a checkbox for 'Enable published model', a text input field with placeholder 'Type a test utterance & press Enter', and a large empty result area. At the top right are buttons for 'Labels view (Ctrl+E)', 'Entities', and 'Reset console'.

- **The test view**, on the left side of the screen, where you can type the test utterance in the text box and press Enter to submit it to your app.
- **The result view**, on the right side of the screen, where your LUIS app returns the test result; i.e. the predicted interpretation of the utterance.

In an interactive test, you submit individual test utterances and view the returned result for each utterance separately.

#### To perform interactive testing on the current model:

- On the **Test App** page, **Interactive Testing** tab, type "book me a flight to Boston tomorrow" as your test utterance in the text box and press Enter. You'll get the following result:

The screenshot shows the LUIS Test App interface with the 'Interactive Testing' tab selected. The test utterance 'book me [number] flight to [\$Location::ToLocation] [\$datetime]' is entered into the text input field. The result panel on the right displays the 'Current version results' section, which includes the 'Top scoring intent' 'Bookflight (0.99)' and 'Other intents' 'GetWeather (0.06) None (0.05)'. The interface also includes a 'Labels view (Ctrl+E)', 'Entities', and 'Reset console' button at the top right.

The testing result includes the top scoring intent identified in the utterance, with its certainty score, as well as other intents existing in your model with their certainty scores. The identified entities will also be displayed within the utterance and you can control their view by selecting your preferred view from the **Labels view** list at the top of the test console.

#### To perform interactive testing on current and published models:

1. On the **Test App** page, **Interactive Testing** tab, click **Enable published model** check box and then click **Yes** in the following confirmation message:

## Enable published model?



Testing utterances hits queried to the published model are deducted from your key quota. Continue?

Yes

No

### NOTE

If you do not have a published version of your application, the **Enable published model** check box will be disabled.

2. Type "book me a flight to Boston tomorrow" as your test utterance and press Enter. The result view on the right side will be split horizontally into two parts (as in the following screenshot) to display results of the test utterance in both the current and published models.

Interactive Testing   Batch Testing

Enable published model

Type a test utterance & press Enter →

book me [\$number] flight to [\$Location::ToLocation]  
[\$datetime]

Labels view (Ctrl+E) Entities Reset console

Current version results

Top scoring intent  
Bookflight (1)

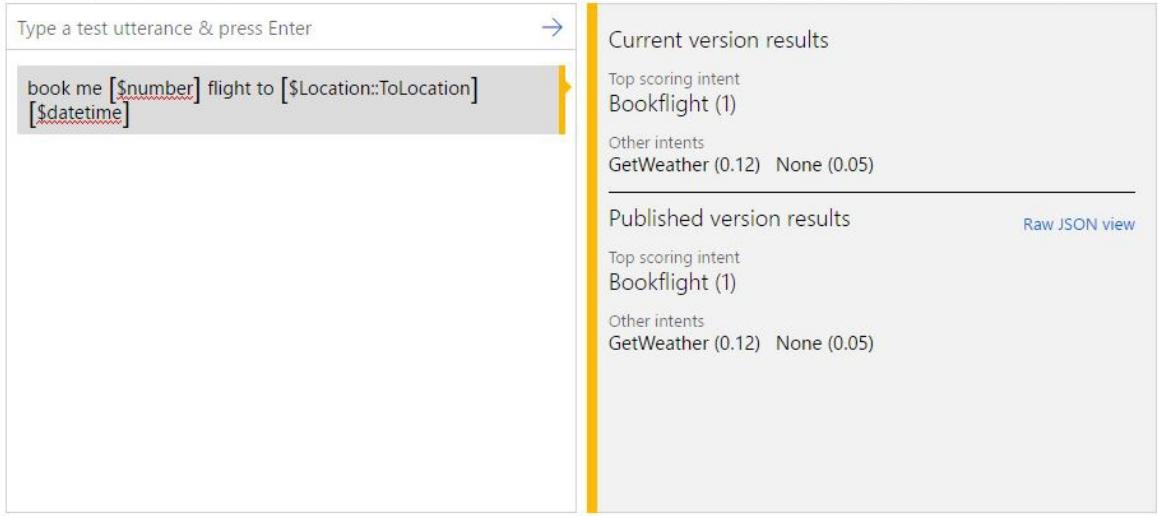
Other intents  
GetWeather (0.12) None (0.05)

Published version results

Raw JSON view

Top scoring intent  
Bookflight (1)

Other intents  
GetWeather (0.12) None (0.05)



3. To view the test result of your published app in JSON format, click **Raw JSON view**. This will look like the following screenshot.

Interactive Testing   Batch Testing

Enable published model

Type a test utterance & press Enter →

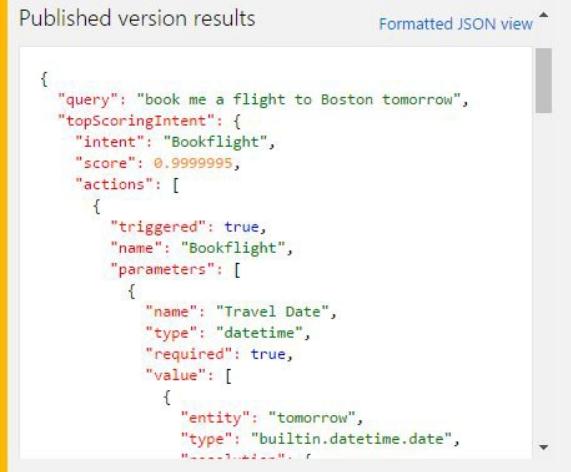
book me [\$number] flight to [\$Location::ToLocation]  
[\$datetime]

Labels view (Ctrl+E) Entities Reset console

Published version results

Formatted JSON view

```
{ "query": "book me a flight to Boston tomorrow", "topScoringIntent": { "intent": "Bookflight", "score": 0.999995, "actions": [ { "triggered": true, "name": "Bookflight", "parameters": [ { "name": "Travel Date", "type": "datetime", "required": true, "value": [ { "entity": "tomorrow", "type": "builtin.datetime.date", "value": "2023-09-25T00:00:00Z" } ] } ] } ] }
```



In case of interactive testing on both trained and published models together, an entity may have a different prediction in each model. In the test result, this entity will be distinguished by a red underline. If you hover over the underlined entity, you can view the entity prediction in both trained and published models.

The screenshot shows the LUIS Interactive Testing interface. On the left, there is a text input field with the placeholder "Type a test utterance & press Enter". Below it, a dropdown menu displays the utterance "book me [\$number] flight to [\$Location::ToLocation] [\$datetime]" with the entity prediction "Entity prediction". Underneath, two entries are shown: "Trained: \$number" and "Published: \$builtin.number". On the right, there are two sections: "Current version results" and "Published version results". The "Current version results" section shows the top scoring intent "Bookflight (1)" and other intents "GetWeather (0.12)" and "None (0.05)". The "Published version results" section shows a JSON response with the query "book me a flight to Boston tomorrow", the top scoring intent "Bookflight", and actions triggered for the Bookflight intent. A "Formatted JSON view" link is also present.

#### NOTE

About the interactive testing console:

- You can type as many test utterances as you want in the test view; only one utterance at a time.
- The result view shows the result of the latest utterance.
- To review the result of a previous utterance, just click it in the test view and its result will be displayed on the right.
- To clear all the entered test utterances and their results from the test console, click **Reset Console** on the top right corner of the console.

## Batch Testing

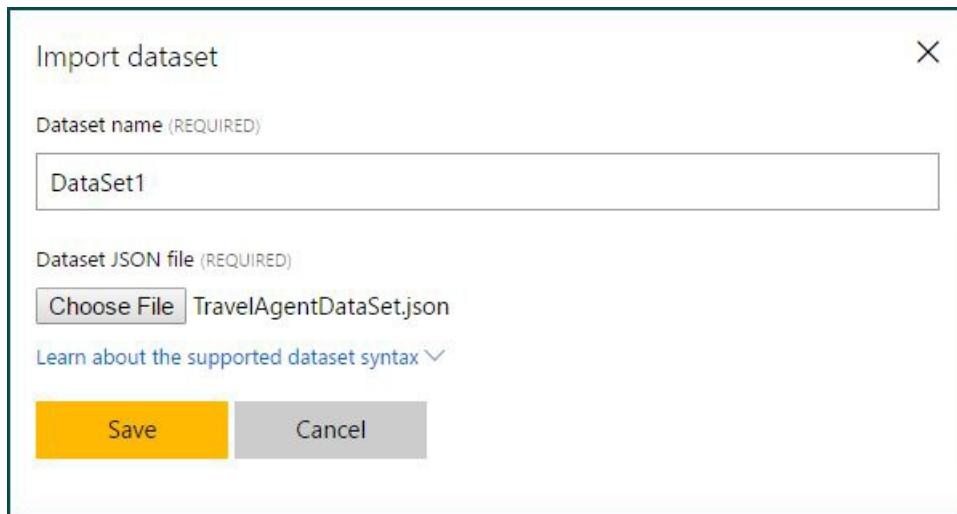
Batch testing enables you to run a comprehensive test on your current trained model to measure its performance in language understanding. In batch testing, you submit a large number of test utterances collectively in a batch file, known as "dataset". The dataset file should be written in JSON format and contains a maximum of 1000 utterances. All you need to do is to import this file to your app and run it to perform the test. Your LUIS app will return the result, enabling you to access detailed analysis of all utterances included in the batch.

You can import up to 10 dataset files to a single LUIS app. It is recommended that the utterances included in the dataset should be different from the example utterances you previously added while building your app.

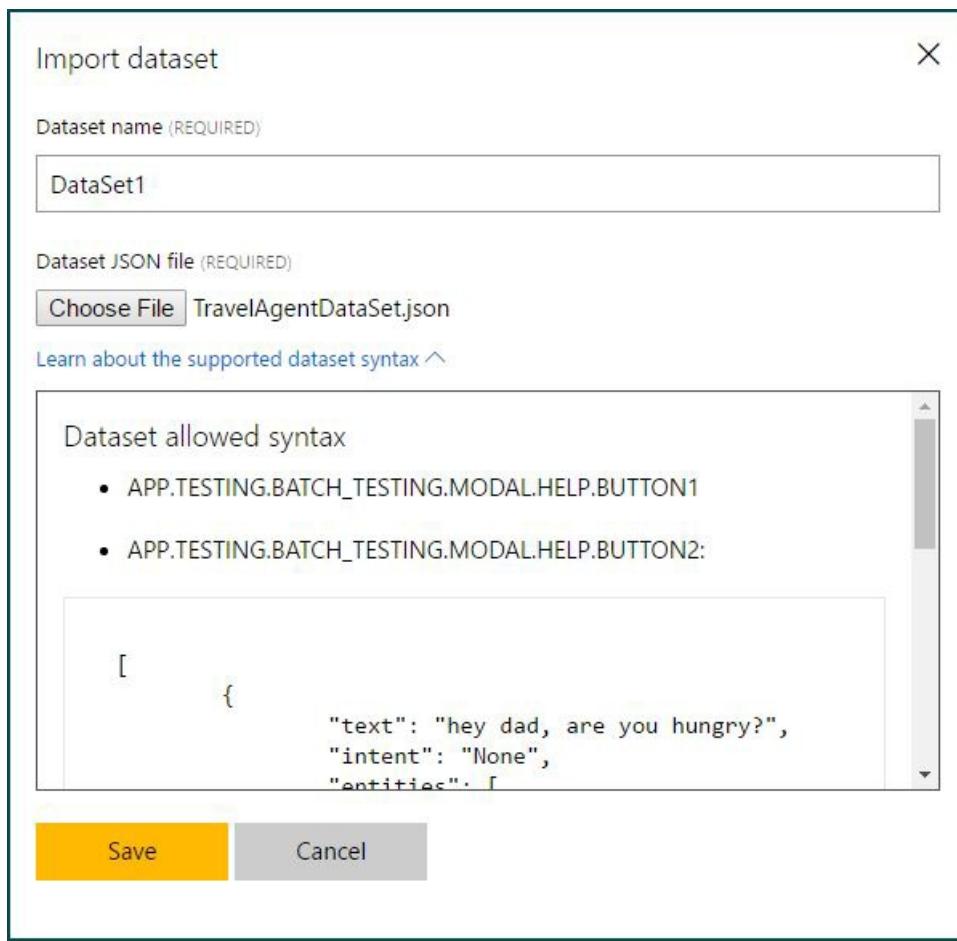
The following procedures will guide you on how to import a dataset file, run a batch test on your current trained app using the imported dataset, and finally to access the test results in a detailed visualized view.

### To import a dataset file:

1. On the **Test App** page, click **Batch Testing**, and then click **Import dataset**. The **Import dataset** dialog box appears.



2. In **Dataset name**, type a name for your dataset file (For example "DataSet1").
3. To learn more about the supported syntax for dataset files to be imported, click **learn about the supported dataset syntax link**. The **Import dataset** dialog box will be expanded displaying the allowed syntax. To collapse the dialog and hide syntax, just click the link again.



4. Click **Choose File** to choose the dataset file you want to import, and then click **Save**. The dataset file will be added.

## Interactive Testing    Batch Testing

**Import dataset**

Status	Name	Utterance Count	Last Test Date	Last Test Success	Controls
▷ Test	DataSet1	9	Not tested yet	Not tested yet	  

5. To rename, delete or download the imported dataset, you can use these buttons respectively: **Rename Dataset** , **Delete Dataset**  and **Download Dataset JSON** .

### To run a batch test on your trained app:

- Click **Test** next to the dataset you've just imported. Soon, the test result of the dataset will be displayed.

## Interactive Testing    Batch Testing

**Import dataset**

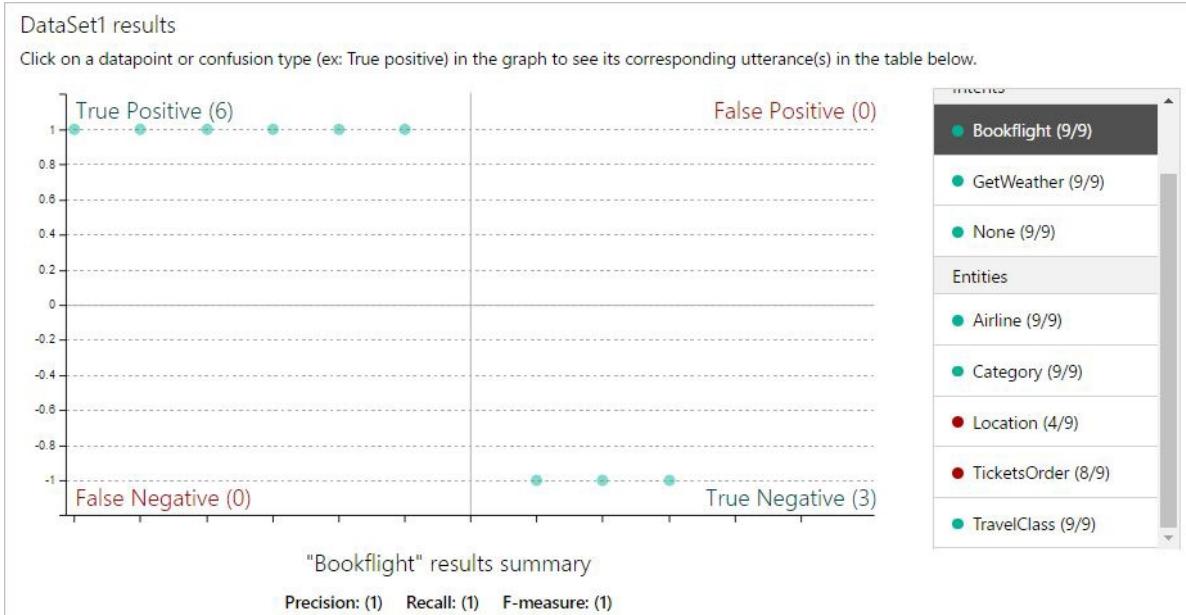
Status	Name	Utterance Count	Last Test Date	Last Test Success	Controls
 See results	DataSet1	9	Feb 23, 2017, 3:36:59 PM	33%	  

In the above screenshot:

- Status** of the dataset shows whether or not the dataset result contains errors. In the above example, an error sign is displayed indicating that there are errors in one or more utterances. If the test result contains no errors, a green sign will be displayed instead.
- Utterance Count** is the total number of utterances included in the dataset file.
- Last Test Date** is the date of the latest test run for this dataset.
- Last Test Success** displays the percentage of correct predictions resulting from the test.

### To access test result details in a visualized view:

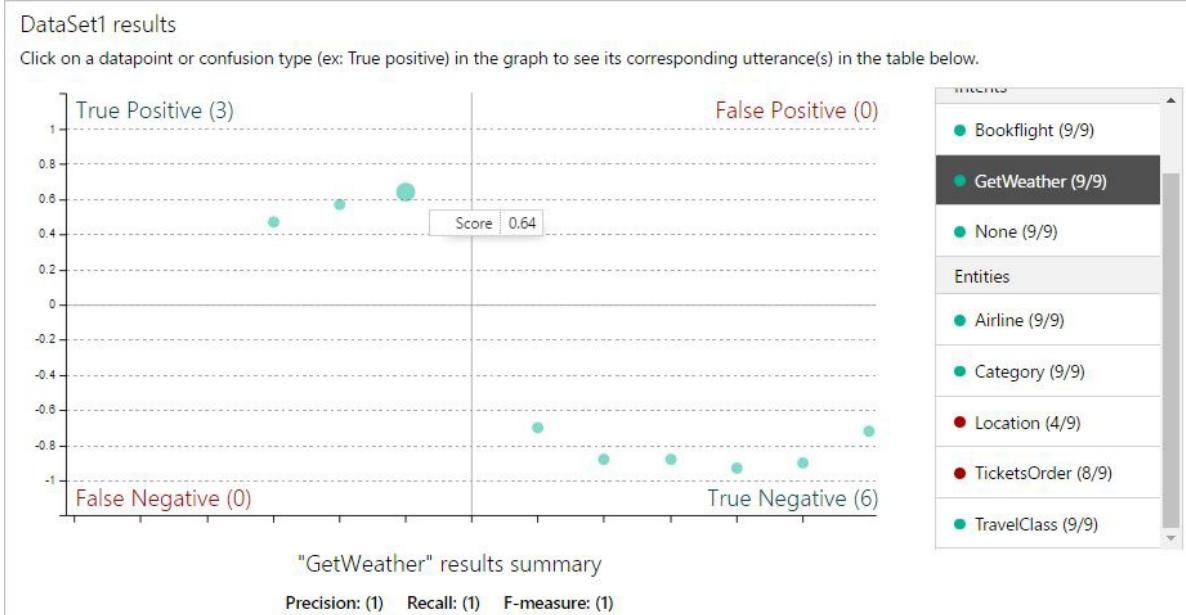
- Click the **See results** link that appears as a result of running the test (see the above screenshot). A scatter graph (confusion matrix) is displayed, where the data points represent the utterances in the dataset. Green points indicate correct prediction and red ones indicate incorrect prediction.



#### NOTE

The filtering panel on the right side of the screen displays a list of all intents and entities in the app, with a green point for intents/entities which were predicted correctly in all dataset utterances, and a red one for those with errors. Also, for each intent/entity, you can see the number of correct predictions out of the total utterances. For example, in the above screenshot, the entity "Location (4/9)" has 4 correct predictions out of 9, so it has 5 errors.

2. To filter the view by a specific intent/entity, click on your target intent/entity in the filtering panel. The data points and their distribution will be updated according to your selection. For example, the following screenshot displays results for the "GetWeather" intent.



#### NOTE

Hovering over a data point shows the certainty score of its prediction.

The graph contains 4 sections representing the possible cases of your application's prediction:

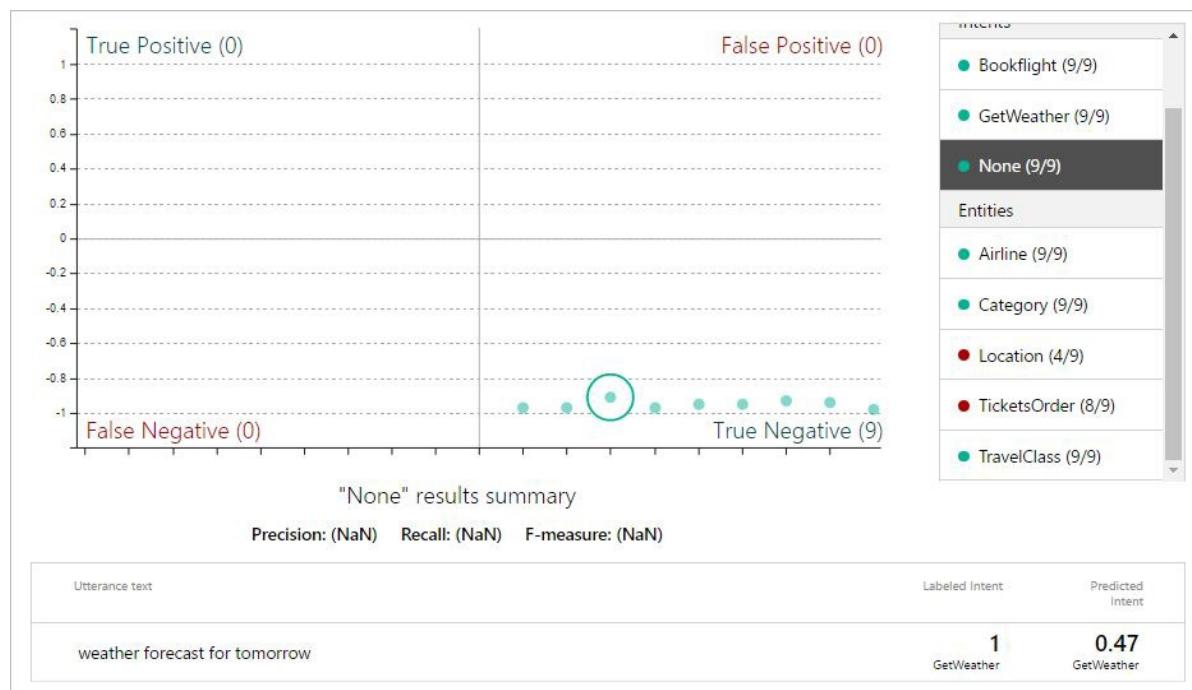
- **True Positive (TP):** The data points in this section represent utterances in which your app correctly predicted the existence of the target intent/entity.

- **True Negative (TN):** The data points in this section represent utterances in which your app correctly predicted the absence of the target intent/entity.
- **False Positive (FP):** The data points in this section represent utterances in which your app incorrectly predicted the existence of the target intent/entity.
- **False Negative (FN):** The data points in this section represent utterances in which your app incorrectly predicted the absence of the target intent/entity.

This means that data points on the **False Positive & False Negative** sections indicate errors, which should be investigated. On the other hand, if all data points are on the **True Positive** and **True Negative** sections, then your application's performance is perfect on this dataset.

3. Click a data point to retrieve its corresponding utterance in the utterances table at the bottom of the page. To display all utterances in a section, click the section title (e.g. True Positive, False Negative, ..etc.)

For example, the following screenshot shows the results for the "None" intent when one of its data points is clicked, so the utterance "weather forecast for tomorrow" is displayed. This utterance falls under the **True Negative** section as your app correctly predicted that the "None" intent is not present in this utterance.



Thus, a batch test helps you view the performance of each intent and entity in your current trained model on a specific set of utterances. This helps you take appropriate actions, when required, to improve performance, such as adding more example utterances to an intent if your app frequently fails to identify it.

# Label Suggested Utterances

4/12/2017 • 3 min to read • [Edit Online](#)

## Suggested utterances demonstrate active machine learning

The breakthrough feature of LUIS is active learning. Once your application is deployed and traffic starts to flow into the system, LUIS uses active learning to improve itself. In the active learning process, LUIS examines all the utterances that have been sent to it, and calls to your attention the ones that it would like you to label. LUIS identifies the utterances that it is relatively unsure of and asks you to label them. Suggested utterances are the utterances that your LUIS app suggests for labeling. If you label these utterances, this will give your application the biggest boost in performance.

## View suggested utterances

Suggested utterances are taken from end-user queries on the application's HTTP endpoint. If your app is not published or has not received hits yet, you will not have any suggested utterances. Also, you will not get suggested utterances for an intent/entity if no endpoint hits are received on this intent/entity.

You can view suggested utterances per intent or entity, under the **Suggested Utterances** tab on the intent page or on the **Entities** page.

### To view suggested utterances per intent:

1. Open your app (e.g. TravelAgent) by clicking its name on **My Apps** page, and then click **Intents** on the app's left panel.
2. On the **Intents** page, click the intent you want to view its suggested utterances (e.g. "Bookflight").
3. On the "Bookflight" intent page, click **Suggested Utterances**. The list of suggested utterances will be displayed. This will look like the following screenshot.

The screenshot shows the LUIS web interface. At the top, there is a navigation bar with links for Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. Below the navigation bar, the main header says "TravelAgent" and "Bookflight". A sub-header below the main header says "Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... Learn more". There are three tabs at the top of the main content area: "Utterances (6)", "Entities in use (2)", and "Suggested utterances". The "Suggested utterances" tab is selected. Below the tabs, there is a toolbar with "Save", "Discard", "Delete", "Reassign Intent", "Labels view (Ctrl+E)", a dropdown menu set to "Entities", and a search bar. The main content area displays a table of suggested utterances. The table has columns for "Utterance text", "Suggested Intent", and a count. There are two rows in the table:

- reserve { [ \$ number ] adult economy } tickets to london
- book me [\$number] flight to [\$Location::ToLocation] [\$datetime]

Both rows have a count of 1 and are labeled "Bookflight".

### To view suggested utterances per entity:

1. Open your app (e.g. TravelAgent) by clicking its name on **My Apps** page, and then click **Entities** on the app's left panel.
2. On the **Entities** page, click **Suggested Utterances**.

The screenshot shows the Microsoft Cognitive Services Language Understanding Entities page. The left sidebar has a 'Entities' section selected. The main area displays a form for managing entities, with tabs for 'Entities list', 'Labeled utterances', and 'Suggested utterances'. A dropdown menu 'Suggest utterances for:' is set to 'Choose Entity ...'. Below it are buttons for 'Save', 'Discard', 'Delete', and 'Reassign Intent'. A 'Labels view (Ctrl+E)' dropdown is set to 'Entities'. A search bar 'Search in utterances ...' is also present. The 'Suggested utterances' section lists three items:

- reserve ([ \$ number ] adult economy ) tickets to london
- book me [\$number] flight to [\$Location::ToLocation] [\$datetime]

3. Choose the entity you want to view its suggested utterances. The list of suggested utterances will be displayed. This will look like the following screenshot.

This screenshot is identical to the one above, showing the 'Suggested utterances' list for the 'Location' entity. It lists the same two suggested utterances with their respective scores and intent types.

Suggested utterances per intent/entity are listed under the **Suggested Utterances** tab. For each suggested utterance, the most likely intent and its score (according to your app's prediction) are displayed. To sort the suggested utterances based on their prediction score, in ascending or descending order, you can click the **Suggested Intent** column header.

To control how you see the words classified as entities in suggested utterances, select a view from the **Labels View** list at the top of the suggested utterances list. You can also press Ctrl+E to quickly switch between views. These are the available views:

- **Tokens:** show entity-classified words in text format.
- **Entities:** show entity-classified words in a tagged format (entity labels enclosed in square brackets). This is the default view.
- **Composite entities:** show the words classified as composite entities in their entity labels.

To filter suggested utterances, click the filter button to display all filters, and then click the filter(s) that you want to apply. For the **Entity** filter, select the entity by which you want to filter the suggested utterances.

## Label suggested utterances

After reviewing the utterances and their predictions, you are advised to take action to label them. You can accept the prediction if it is correct and save the utterance as is, or make any required changes to ensure they are correctly labeled. For example, you can change the intent of a suggested utterance (reassign intent), label unlabeled entities,

correct mis-labeled ones or even remove their labels.

The following are the possible cases you may have, along with the actions you can take in each case in order to label the suggested utterances (using examples from the TravelAgent app):

- If the predicted intent is correct and entities are correctly detected and labeled, then just select the utterance and click **Save** without making any changes to accept and save the utterance.
- If the predicted intent is incorrect, you need to change it. For example, in the screenshot below note that the predicted intent in the last suggested utterance "reserve a ticket to London" is "GetWeather", which is incorrect. Click **Reassign Intent** and choose the correct intent "Bookflight" from the list.

## Bookflight

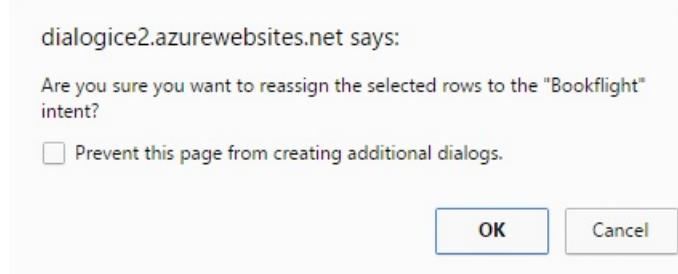
Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (4) Entities in use (1) Suggested utterances

Review the suggested utterances of the current intent and make any required changes to ensure they are correctly labeled ... [Learn more](#)

The screenshot shows a 'Reassign Intent' dialog for the 'Bookflight' intent. The dialog has tabs for 'Save' and 'Discard'. The main area shows a list of utterances with their predicted intents and confidence scores. The utterance 'reserve a ticket to [\$Location::ToLocation]' is selected and highlighted in green. A tooltip 'Reassign intent for selected utterance(s)' appears over the 'Reassign Intent' button. The 'Suggested Intent' dropdown shows 'Bookflight' with a confidence of 0.67. Other listed intents include 'GetWeather (0.42)', 'Bookflight (0.35)', 'None (0.13)', and 'Bookflight (0.42)'. The bottom right corner of the dialog has a green box containing the number '1'.

You will get a confirmation message. Click **OK** to confirm this action, and then click **Save** to save your changes.



- If a phrase is unlabeled, click it and select an entity label from the list. For example, "KSA" in the screenshot below is unlabeled. Click it and select "To Location" as its entity label and then click **Save**.

## Entities

Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)

Entities list Labeled utterances Suggested utterances

Retrieve suggested utterances that contain a specific entity and make any required changes to improve your model ... [Learn more](#)

Suggest utterances for :

The screenshot shows the 'Entities' list interface. At the top, there are buttons for Save, Discard, and Reassign Intent. A dropdown menu labeled 'Labels view (Ctrl+E): Entities' is open. Below the menu, there is a search bar 'Search in utterances ...'. The main area displays a table of utterances. One row has a checked checkbox next to 'Utterance text' and a 'Suggested Intent' column showing 'Bookflight' with a score of '0.67'. Another row shows 'i want to travel to [ksa]' with a score of '0.42'. A third row has a checked checkbox next to 'reserve a ticket to [\$L...' and a 'Suggested Intent' column showing 'GetWeather' with a score of '0.42'. A modal dialog box is overlaid on the table, containing a search input 'Search/Add entity ...' and a dropdown menu with options: 'Back', 'Location::FromLocation', and 'Location::ToLocation'. The 'Location::ToLocation' option is highlighted.

Utterance text	Suggested Intent	Score
book me a flight to [\$Location::ToLocation] on [\$datetime][\$number]	Bookflight	0.67
i want to travel to [ksa]	Bookflight	0.42
reserve a ticket to [\$L...	GetWeather	0.42

- If an entity is mislabeled, click it and then you can either select the correct label or click **Remove Label** to remove its label. Then, click **Save**.

[My Apps](#) > [TravelAgent](#) > [Intents](#) > Bookflight

## Bookflight

Here you are in full control of this intent: you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (6) Entities in use (2) Suggested utterances

The screenshot shows the 'Bookflight' intent management interface. At the top, there are buttons for Save, Discard, and Reassign Intent. A dropdown menu labeled 'Labels view (Ctrl+E): Entities' is open. Below the menu, there is a search bar 'Search in utterances ...'. The main area displays a table of utterances. One row shows 'book me a flight to [\$Location::ToLocation] on may [\$number]' with a score of '0.99'. Another row shows 'i want to travel to ksa' with a score of '0.89'. A third row shows 'reserve a ticket to [\$Location::ToLocation]' with a score of '0.97'. A modal dialog box is overlaid on the table, containing a search input 'Search/Add entity ...' and a dropdown menu with options: 'Remove label', 'Airline', 'TicketsOrder (Composite)', 'Location (Hierarchical)', 'TravelClass (Hierarchical)', and 'Category (Hierarchical)'. The 'Location (Hierarchical)' option is highlighted.

Utterance text	Suggested Intent	Score
book me a flight to [\$Location::ToLocation] on may [\$number]	Bookflight	0.99
i want to travel to ksa	Bookflight	0.89
reserve a ticket to [\$Location::ToLocation]	Bookflight	0.97

# Publish your app

4/18/2017 • 2 min to read • [Edit Online](#)

When you finish building and testing your app, you can publish it as webservice on Azure and get an HTTP endpoint that can be integrated in any backend code.

Optionally, it would be better to test your app before publishing it. For instructions, see [Train and test your app](#).

You can either publish your app directly to the **Production Slot** where end users can access and use your model, or you can publish to a **Staging Slot** to use as a development workstation where you can work on your app and go through trials and tests to validate any changes before publishing to the production slot.

## To publish your app to an HTTP endpoint:

1. Open your app (e.g. TravelAgent) by clicking its name on **My Apps** page, and then click **Publish App** in the left panel to access the **Publish App** page. The screenshot below shows the Publish page if you haven't published your app yet.

The screenshot shows the 'Publish App' page for the 'TravelAgent' application. The left sidebar lists navigation options: Dashboard, Intents, Entities, Features, Train & Test, and Publish App (which is currently selected). Below the sidebar, a link leads back to the 'App list'. The main content area is titled 'Publish App' and contains the following sections:

- Essentials**: A message stating 'Latest publish: You haven't published your application yet'. Below this is a dropdown menu labeled 'Endpoint Key (REQUIRED)' with the placeholder 'Choose a key to assign to application ...' and a link 'Add a new key to your account'.
- Publish settings**: A dropdown menu labeled 'Endpoint slot' set to 'Production'. A note below states 'This slot has no published application.'
- External Key Associations**: A button labeled 'Add Key Association'.

At the bottom of the page, there are 'Train' and 'Publish' buttons, with a note 'Please train your app to publish.' positioned between them.

If you have previously published this app, this page will look like the following screenshot:

The screenshot shows the LUIS Publish App interface. On the left, there's a sidebar with options like Dashboard, Intents, Entities, Features, Train & Test, and Publish App. The Publish App section is active. In the main area, it says "Publish App" and provides instructions to publish your app as a web service or chat bot. It shows the latest publish was 9 hours ago. A dropdown for "Endpoint Key (REQUIRED)" is set to "Tier1". Below it, a link says "Add a new key to your account". Under "Publish settings", the "Endpoint slot" dropdown is set to "Production". "Slot info" shows the published version ID is 0.1 and the date is Feb 28, 2017, 2:07:53 PM. The "Endpoint url" is displayed as a long URL starting with https://dialogice3.cloudapp.net:8081/api/v2.0/apps/54a3e66c-b587-45f7-b58c-dcb8f7505373?subscription-key=01234567890123456789012345678998&q=. There are checkboxes for "Add verbose flag" and "Enable bing spell checker". At the bottom, there are "Train" and "Publish" buttons, with "Please train your app to publish." in red text above the "Train" button. The "External Key Associations" section has a yellow "Add Key Association" button.

2. From the **Endpoint Key** list, select an existing endpoint key to assign to the app, or click **Add a new key to your account** to add a new one. For more information on how to create and add endpoint keys to your account, see [Manage your keys](#).
3. Choose whether to publish to **Production** or to **Staging** slot by selecting the corresponding value from the **Endpoint Slot** list.
4. If you will use an external service with your LUIS app (e.g. Bing Spell Check):
  - Click **Add Key Association** to assign the external service key to the app by selecting the key type and key value in the following dialog box.
  - Click **Enable Bing Spell Checker** check box.

The dialog box is titled "Add Key Association" and has an "X" button in the top right corner. It contains two input fields: "Key Type (REQUIRED)" which is set to "BingSpellCheck", and "Key Value (REQUIRED)" which is set to "01234567890123456789012345678901". At the bottom are two buttons: a yellow "Save" button and a grey "Cancel" button.

5. If you want the JSON response of your published app to include all intents defined in your app and their prediction scores, click **Add Verbose Flag** checkbox. Otherwise, it will include only the top scoring intent.
6. Click **Train** if the app wasn't trained already.
7. Click **Publish**. The endpoint URL of your published app is displayed.

Publish settings

Endpoint slot

Production

Slot info

Published version Id: 0.1 Published date: Feb 28, 2017, 11:48:49 PM

Endpoint url

[https://dialogice3.cloudapp.net:8081/api/v2.0/apps/54a3e66c-b587-45f7-b58c-dcb8f7505373?subscription-key=01234567890123456789012345678998&q=Book me a flight to Boston on May 4](https://dialogice3.cloudapp.net:8081/api/v2.0/apps/54a3e66c-b587-45f7-b58c-dcb8f7505373?subscription-key=01234567890123456789012345678998&q=Book%20me%20a%20flight%20to%20Boston%20on%20May%204)

Add verbose flag  Enable bing spell checker

Train Publish

**NOTE**

If the **Publish** button is disabled, then either your app does not have an assigned an endpoint key, or you have not trained your app yet.

To test how your published app works, you can access the test console by clicking **Train & Test** in the left panel. For instructions on how to test your app using the test console, see [Train and test your app](#).

If you want to test your published endpoint in a browser using the generated URL, you can click the URL to open it in your browser, then set the URL parameter "&q" to your test query (for example: "&q=Book me a flight to Boston on May 4"), and then press Enter. You will get the JSON response of your HTTP endpoint.

```
{  
  "query": "Book me a flight to Boston on May 4",  
  "topScoringIntent": {  
    "intent": "Bookflight",  
    "score": 0.610224  
  },  
  "entities": [  
    {  
      "entity": "may 4",  
      "type": "builtin.datetime.date",  
      "startIndex": 30,  
      "endIndex": 34,  
      "resolution": {  
        "date": "XXXX-05-04"  
      }  
    },  
    {  
      "entity": "4",  
      "type": "builtin.number",  
      "startIndex": 34,  
      "endIndex": 34,  
      "score": 0.9979208  
    }  
  ]  
}
```

# Pre-built Entities

4/12/2017 • 6 min to read • [Edit Online](#)

LUIS includes a set of pre-built entities. When a pre-built entity is included in your application, its predictions will be included in your published application and be available to you in the LUIS web UI while labeling utterances. The behavior of pre-built entities **cannot** be modified. Unless otherwise noted, pre-built entities are available in all LUIS application locales (cultures). Below is a table of pre-built entities supported per culture.

PRE-BUILT ENTITY	EN-US	FR-FR	IT-IT	ES-ES	ZH-CN	DE-DE	PT-BR	JA-JP	KO-KR
Datetime	<input checked="" type="checkbox"/>	-							
Number	<input checked="" type="checkbox"/>	-							
Ordinal	<input checked="" type="checkbox"/>	-							
Percentage	<input checked="" type="checkbox"/>	-							
Temperature	<input checked="" type="checkbox"/>	-							
Dimension	<input checked="" type="checkbox"/>	-							
Money	<input checked="" type="checkbox"/>	-							
Age	<input checked="" type="checkbox"/>	-							
Geography	<input checked="" type="checkbox"/>	-	-	-	-	-	-	-	-
Encyclopedia	<input checked="" type="checkbox"/>	-	-	-	-	-	-	-	-
URL	<input checked="" type="checkbox"/>	-	-	-	-	-	-	-	-
Email	<input checked="" type="checkbox"/>	-	-	-	-	-	-	-	-
Phone number	<input checked="" type="checkbox"/>	-	-	-	-	-	-	-	-

Below is a table of pre-built entities with example utterances and their return values.

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.number	ten	{ "type": "builtin.number", "entity": "ten" }

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.number	3.1415	{ "type": "builtin.number", "entity": "3 . 1415" }
builtin.ordinal	first	{ "type": "builtin.ordinal", "entity": "first" }
builtin.ordinal	10th	{ "type": "builtin.ordinal", "entity": "10th" }
builtin.temperature	10 degrees celcius	{ "type": "builtin.temperature", "entity": "10 degrees celcius" }
builtin.temperature	78 F	{ "type": "builtin.temperature", "entity": "78 f" }
builtin.dimension	2 miles	{ "type": "builtin.dimension", "entity": "2 miles" }
builtin.dimension	650 square kilometers	{ "type": "builtin.dimension", "entity": "650 square kilometers" }
builtin.money	1000.00 US dollars	{ "type": "builtin.money", "entity": "1000.00 us dollars" }
builtin.money	\$ 67.5 B	{ "type": "builtin.money", "entity": "\$ 67.5" }
builtin.age	100 year old	{ "type": "builtin.age", "entity": "100 year old" }
builtin.age	19 years old	{ "type": "builtin.age", "entity": "19 years old" }
builtin.percentage	The stock price increase by 7 \$ this year	{ "type": "builtin.percentage", "entity": "7 %" }
builtin.datetime	See separate table	See separate table below
builtin.geography	See separate table	See separate table below
builtin.encyclopedia	See separate table	See separate table below

The last 3 built-in entity types listed in the table above encompass multiple subtypes. These are covered next.

### **builtin.datetime**

The builtin.datetime pre-built entity has awareness of the current date and time. In the examples below, the current date is 2015-08-14. Also, builtin.datetime provides a resolution field that produces a machine-readable dictionary.

**This pre-built entity has 3 subtypes:**

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
------------------	-------------------	------

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetime.date	tomorrow	{ "type": "builtin.datetime.date", "entity": "tomorrow", "resolution": {"date": "2015-08-15"} }
builtin.datetime.date	january 10 2009	{ "type": "builtin.datetime.date", "entity": "january 10 2009", "resolution": {"date": "2009-01-10"} }
builtin.datetime.date	monday	{ "entity": "monday", "type": "builtin.datetime.date", "resolution": {"date": "XXXX-WXX-1"} }
builtin.datetime.date	next week	{ "entity": "next week", "type": "builtin.datetime.date", "resolution": {"date": "2015-W34"} }
builtin.datetime.date	next monday	{ "entity": "next monday", "type": "builtin.datetime.date", "resolution": {"date": "2015-08-17"} }
builtin.datetime.date	week of september 30th	{ "entity": "week of september 30th", "type": "builtin.datetime.date", "resolution": {"comment": "weekof", "date": "XXXX-09-30"} }
builtin.datetime.time	3 : 00	{ "type": "builtin.datetime.time", "entity": "3 : 00", "resolution": {"comment": "ampm", "time": "T03:00"} }
builtin.datetime.time	4 pm	{ "type": "builtin.datetime.time", "entity": "4 pm", "resolution": {"time": "T16"} }
builtin.datetime.time	tomorrow morning	{ "entity": "tomorrow morning", "type": "builtin.datetime.time", "resolution": {"time": "2015-08-15TMO"} }
builtin.datetime.time	tonight	{ "entity": "tonight", "type": "builtin.datetime.time", "resolution": {"time": "2015-08-14TNI"} }
builtin.datetime.duration	for 3 hours	{ "type": "builtin.datetime.duration", "entity": "3 hours", "resolution": {"duration": "PT3H"} }
builtin.datetime.duration	30 minutes long	{ "type": "builtin.datetime.duration", "entity": "30 minutes", "resolution": {"duration": "PT30M"} }

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetime.duration	all day	{ "type": "builtin.datetime.duration", "entity": "all day", "resolution": {"duration": "P1D"} }
builtin.datetime.set	daily	{ "type": "builtin.datetime.set", "entity": "daily", {"resolution": "time": "XXXX-XX-XX"} }
builtin.datetime.set	every morning	{ "type": "builtin.datetime.set", "entity": "every morning", "resolution": {"time": "XXXX-XX-XXTMO"} }
builtin.datetime.set	every tuesday	{ "entity": "every tuesday", "type": "builtin.datetime.set", "resolution": {"time": "XXXX-WXX-2"} }
builtin.datetime.set	every week	{ "entity": "every week", "type": "builtin.datetime.set", "resolution": {"time": "XXXX-WXX"} }

## builtin.geography

Note: builtin.geography is available only in en-us.

The **builtin.geography** built-in entity type has 3 sub-types:

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.geography.city	seattle	{ "type": "builtin.geography.city", "entity": "seattle" }
builtin.geography.city	paris	{ "type": "builtin.geography.city", "entity": "paris" }
builtin.geography.country	australia	{ "type": "builtin.geography.country", "entity": "australia" }
builtin.geography.country	japan	{ "type": "builtin.geography.country", "entity": "japan" }
builtin.geography.pointOfInterest	amazon river	{ "type": "builtin.geography.pointOfInterest", "entity": "amazon river" }
builtin.geography.pointOfInterest	sahara desert	{ "type": "builtin.geography.pointOfInterest", "entity": "sahara desert" }

## builtin.encyclopedia

Note: builtin.encyclopedia is available only in en-US.

The **builtin.encyclopedia** built-in entity includes over 100 sub-types, listed below. In addition, encyclopedia entities often map to multiple types. For example, the query Ronald Reagan yields:

```
{
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.people.person"
},
{
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.film.actor"
},
{
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.government.us_president"
},
{
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.book.author"
}
```

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.people.person	bryan adams
builtin.encyclopedia.people.person	builtin.encyclopedia.film.producer	walt disney
builtin.encyclopedia.people.person	builtin.encyclopedia.film.cinematographer	adam greenberg
builtin.encyclopedia.people.person	builtin.encyclopedia.royalty.monarch	elizabeth ii
builtin.encyclopedia.people.person	builtin.encyclopedia.film.director	steven spielberg
builtin.encyclopedia.people.person	builtin.encyclopedia.film.writer	alfred hitchcock
builtin.encyclopedia.people.person	builtin.encyclopedia.film.actor	robert de niro
builtin.encyclopedia.people.person	builtin.encyclopedia.martial_arts.martial_artist	bruce lee
builtin.encyclopedia.people.person	builtin.encyclopedia.architecture.architect	james gallier
builtin.encyclopedia.people.person	builtin.encyclopedia.geography.mountaineer	jean couzy
builtin.encyclopedia.people.person	builtin.encyclopedia.celebrities.celebrity	angelina jolie
builtin.encyclopedia.people.person	builtin.encyclopedia.music.musician	bob dylan
builtin.encyclopedia.people.person	builtin.encyclopedia.soccer.player	diego maradona
builtin.encyclopedia.people.person	builtin.encyclopedia.baseball.player	babe ruth
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.player	heiko schaffartzik
builtin.encyclopedia.people.person	builtin.encyclopedia.olympics.athlete	andre agassi

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.coach	bob huggins
builtin.encyclopedia.people.person	builtin.encyclopedia.american_football.coach	james franklin
builtin.encyclopedia.people.person	builtin.encyclopedia.cricket.coach	andy flower
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.coach	david quinn
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.player	vincent lecavalier
builtin.encyclopedia.people.person	builtin.encyclopedia.government.politician	harold nicolson
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_president	barack obama
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_vice_president	dick cheney
builtin.encyclopedia.organization.organization	builtin.encyclopedia.organization.organization	united nations
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.league	american league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.conference	western hockey league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.division	american league east
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.league	major league baseball
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.conference	national basketball league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.division	pacific division
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.league	premier league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.division	afc north
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.broadcast	nebraska educational telecommunications
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_station	abc

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_channel	cnbc world
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.radio_station	bbc radio 1
builtin.encyclopedia.organization.organization	builtin.encyclopedia.business.operation	bank of china
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.record_label	pixar
builtin.encyclopedia.organization.organization	builtin.encyclopedia.aviation.airline	air france
builtin.encyclopedia.organization.organization	builtin.encyclopedia.automotive.company	general motors
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.musical_instrument_company	gibson guitar corporation
builtin.encyclopedia.organization.organization	builtin.encyclopedia.tv.network	cartoon network
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.educational_institution	cornwall hill college
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.school	boston arts academy
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.university	johns hopkins university
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.team	united states national handball team
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.team	chicago bulls
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.professional_sports_team	boston celtics
builtin.encyclopedia.organization.organization	builtin.encyclopedia.cricket.team	mumbai indians
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.team	houston astros
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.team	green bay packers
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.team	hamilton bulldogs

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.team	fc bayern munich
builtin.encyclopedia.organization.organization	builtin.encyclopedia.government.political_party	pertubuhan kebangsaan melayu singapura
builtin.encyclopedia.time.event	builtin.encyclopedia.time.event	1740 batavia massacre
builtin.encyclopedia.time.event	builtin.encyclopedia.sports.championship_event	super bowl xxxix
builtin.encyclopedia.time.event	builtin.encyclopedia.award.competition	eurovision song contest 2003
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.series_episode	the magnificent seven
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.multipart_tv_episode	the deadly assassin
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.commerce.consumer_product	nokia lumia 620
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.music.album	dance pool
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.automotive.model	pontiac fiero
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.computer	toshiba satellite
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.web_browser	internet explorer
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.commerce.brand	diet coke
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.automotive.make	chrysler
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.artist	michael jackson
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.group	the yardbirds
builtin.encyclopedia.music.music_video	builtin.encyclopedia.music.music_video	the beatles anthology
builtin.encyclopedia.theater.play	builtin.encyclopedia.theater.play	camelot
builtin.encyclopedia.sports.fight_song	builtin.encyclopedia.sports.fight_song	the cougar song
builtin.encyclopedia.film.series	builtin.encyclopedia.film.series	the twilight saga
builtin.encyclopedia.tv.program	builtin.encyclopedia.tv.program	late night with david letterman

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.radio.radio_program	builtin.encyclopedia.radio.radio_program	grand ole opry
builtin.encyclopedia.film.film	builtin.encyclopedia.film.film	alice in wonderland
builtin.encyclopedia.cricket.tournament	builtin.encyclopedia.cricket.tournament	cricket world cup
builtin.encyclopedia.government.government	builtin.encyclopedia.government.government	european commission
builtin.encyclopedia.sports.team_owner	builtin.encyclopedia.sports.team_owner	bob castellini
builtin.encyclopedia.music.genre	builtin.encyclopedia.music.genre	eastern europe
builtin.encyclopedia.ice_hockey.division	builtin.encyclopedia.ice_hockey.division	hockeyallsvenskan
builtin.encyclopedia.architecture.style	builtin.encyclopedia.architecture.style	spanish colonial revival architecture
builtin.encyclopedia.broadcast.producer	builtin.encyclopedia.broadcast.producer	columbia tristar television
builtin.encyclopedia.book.author	builtin.encyclopedia.book.author	adam maxwell
builtin.encyclopedia.religion.founding_figure	builtin.encyclopedia.religion.founding_figure	gautama buddha
builtin.encyclopedia.martial_arts.martial_art	builtin.encyclopedia.martial_arts.martial_art	american kenpo
builtin.encyclopedia.sports.school	builtin.encyclopedia.sports.school	yale university
builtin.encyclopedia.business.product_line	builtin.encyclopedia.business.product_line	canon powershot
builtin.encyclopedia.internet.website	builtin.encyclopedia.internet.website	bing
builtin.encyclopedia.time.holiday	builtin.encyclopedia.time.holiday	easter
builtin.encyclopedia.food.candy_bar	builtin.encyclopedia.food.candy_bar	cadbury dairy milk
builtin.encyclopedia.finance.stock_exchange	builtin.encyclopedia.finance.stock_exchange	tokyo stock exchange
builtin.encyclopedia.film.festival	builtin.encyclopedia.film.festival	berlin international film festival

# Cortana Prebuilt App

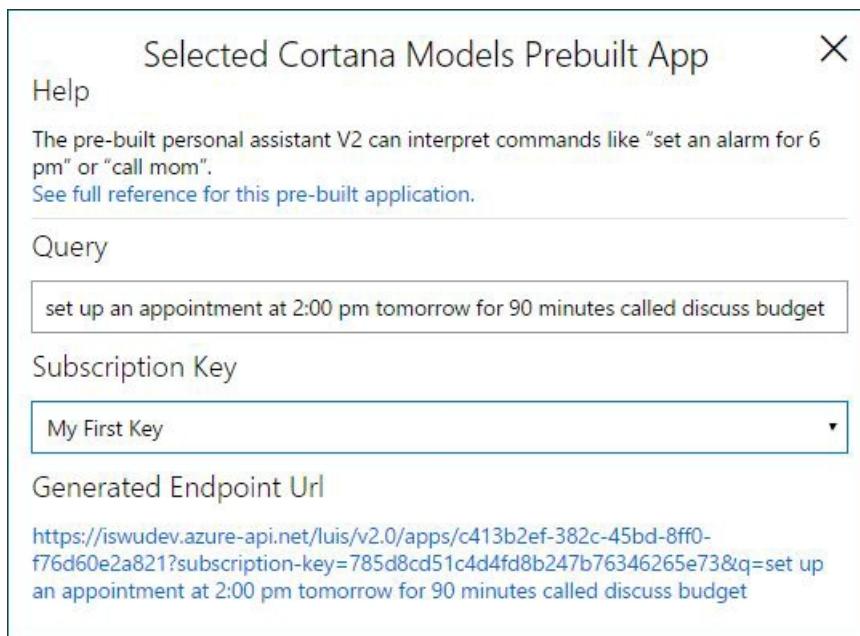
4/12/2017 • 1 min to read • [Edit Online](#)

In addition to allowing you to build your own applications, LUIS also provides selected models from Microsoft Cortana personal assistant as a pre-built application. This is an "as-is" application; the intents and entities in this application cannot be edited or integrated into other LUIS applications. If you'd like your client to have access to both this pre-built application and your own LUIS application, then your client will have to make two separate HTTP calls.

The pre-built personal assistant application is available in these cultures (locales): English, French, Italian, Spanish and Chinese.

## To use Cortana Prebuilt App:

1. On **My Apps** page, click **Cortana prebuilt apps** and select your language, English for example. The following dialog box appears:



2. In **Query**, type the utterance you want to interpret. For example, type "set up an appointment at 2:00 pm tomorrow for 90 minutes called discuss budget"
3. From the **Subscription Key** list, select the subscription key to be used for this endpoint hit to Cortana app.
4. Click the generated endpoint URL to access the endpoint and get the result of the query. The screenshot below shows the result returned in JSON format for the example utterance: "set up an appointment at 2:00 pm tomorrow for 90 minutes called discuss budget"

```
{  
  "query": "set up an appointment at 2:00 pm tomorrow for 90 minutes called discuss budget",  
  "intents": [  
    {  
      "intent": "builtin.intent.calendar.create_calendar_entry"  
    }  
  ],  
  "entities": [  
    {  
      "entity": "2:00 pm",  
      "type": "builtin.calendar.start_time",  
      "resolution": {  
        "resolution_type": "builtin.datetime.time",  
        "time": "T14:00"  
      }  
    },  
    {  
      "entity": "tomorrow",  
      "type": "builtin.calendar.start_date",  
      "resolution": {  
        "date": "2017-02-20",  
        "resolution_type": "builtin.datetime.date"  
      }  
    },  
    {  
      "entity": "90 minutes",  
      "type": "builtin.calendar.duration",  
      "resolution": {  
        "duration": "PT90M",  
        "resolution_type": "builtin.datetime.duration"  
      }  
    },  
    {  
      "entity": "discuss budget",  
      "type": "builtin.calendar.title"  
    }  
  ]  
}
```

# Manage your keys

4/12/2017 • 3 min to read • [Edit Online](#)

A key is your passport to the server allowing you to publish your app to be used by end users. LUIS has three different types of keys:

- **Programmatic API Key:** Created automatically per LUIS account and you don't have to buy it. It enables you to author and edit your application using LUIS programmatic APIs. [Click here for a complete API Reference.](#)
- **Endpoint Key(s):** You need to buy it from Microsoft Azure portal. It is essential for publishing your app and accessing your HTTP endpoint. This key reflects your quota of endpoint hits based on the usage plan you specified while creating the key.
- **External Key(s):** You need to buy an external key only if you want to use any external services with LUIS (e.g. Bing Spell Check service).

The process of creating and using endpoint and external keys involves the following tasks in the same order:

1. Create a key on Azure portal.
2. Add the key to your LUIS account (on **My Keys** page).
3. Assign the key to your app on the **Publish** page.

## Reset Programmatic API key

You can reset your Programmatic API key to get a new one generated for your account.

### To reset a Programmatic API key:

1. Click **My Keys** on LUIS top navigation bar to access **My Keys** page.

The screenshot shows the Microsoft Cognitive Services My Keys page. At the top, there's a navigation bar with the Microsoft logo, 'Cognitive Services', and user information ('Thanaa Al Shamy' and 'Sign out'). Below the navigation bar, the main title is 'My Keys'. A sub-instruction says: 'Here you can set up the keys of your LUIS account: the Programmatic API Key and Azure keys ... [Learn more](#)'. Underneath, the current 'Programmatic API Key' is listed as '785d8cd51c4d4fd8b247b76p46265e73'. A prominent button labeled 'Reset Programmatic API Key' is visible. Below the key, there are two tabs: 'Endpoint Keys' (which is selected) and 'External Keys'. At the bottom, there's a 'Add a new key' button and a table with columns for 'Key Type', 'Key', and 'Actions'.

2. At the top of **My Keys** page, you can see your current Programmatic API key. Click **Reset Programmatic API Key**. The new key will be generated replacing the existing one.

## Create and use endpoint keys for your apps

You can create as many endpoint keys as you need for your LUIS apps on Azure portal. These keys will be used for publishing your apps to the Web.

## To create an endpoint key on Azure portal:

1. Click **My Keys** on LUIS top navigation bar to access **My Keys** page.
2. On **My Keys** page, **Endpoint Keys** tab, click **Buy Key on Azure**. This will take you directly to Microsoft Azure portal. You can create one or more keys per account by subscribing to one or more subscription tiers.
3. For further instructions, see [Creating a subscription key via Azure Ibiza](#).

## To add the endpoint key to your LUIS account:

1. On **My Keys** page, **Endpoint Keys** tab, click **Add a new key**.

Add a new key

Key Value (REQUIRED)

01234567890123456789012345678912

Key Name (OPTIONAL)

Tier1

Save Cancel

2. Copy the key you created in Azure portal in the previous procedure and paste it in the **Key Value** text box.
3. You can optionally type a name for the key in **Key Name**, for example "Tier1", and then click **Save**. The key will be added to the keys list.

My Keys

Here you can set up the keys of your LUIS account: the Programmatic API Key and Azure keys ... [Learn more](#)

Programmatic API Key: 785d8cd51c4d4fd8b247b76p46265e73

Reset Programmatic Api Key

Endpoint Keys External Keys

Key Name	Key	Assigned apps	Actions
Tier1	01234567890123456789012345678998		

In the list of added keys, you can edit a key name, or delete a key (if no longer needed). To do this, click the edit or delete button corresponding to that key in the list.

## To assign the endpoint key to your app:

1. Access the **Publish** page by clicking **Publish App** on the left panel.
2. From the **Endpoint Key** list, select the key that you want to assign to the app.

### NOTE

Whenever you assign a key to the app, an updated endpoint URL will be generated. Remember to use the updated endpoint URL in your code.

## Create and use external keys

External keys are the keys required for any external services that you want to use with your LUIS app to enhance the language understanding experience. One of these services, which is currently available, is [Bing Spell Check](#). It can be used with LUIS to detect and correct any spelling mistakes in end-user queries submitted to your application's endpoint.

**To create and add an external key to your LUIS account:**

1. On **My Keys** page, **External Keys**, click **Add a new key**. The following dialog box appears.

The dialog box is titled "Add a new key". It contains a "Key Type (REQUIRED)" dropdown menu with "BingSpellCheck" selected. Below it is a "Key Value (REQUIRED)" input field with the placeholder "Type Key here ...". At the bottom are two buttons: "Save" and "Cancel".

2. From the **Key Type** list, select "BingSpellCheck".
3. Copy the key you created on Azure and paste it in **Key Value**, and then click **Save**. The key will be added to the keys list.

**To assign the external key to your app:**

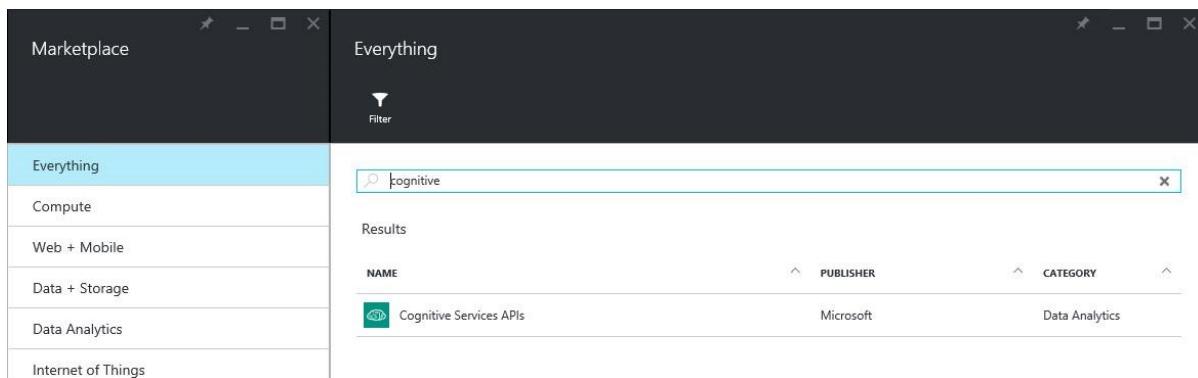
1. Access the **Publish** page by clicking **Publish App** on the left panel.
2. Click **Add Key Association** 
3. Select Bing Spell Check as the key type from the **Key Type** list, and select from the **Key Value** list the external key that you want to assign to the app.

# Creating Subscription Keys Via Azure

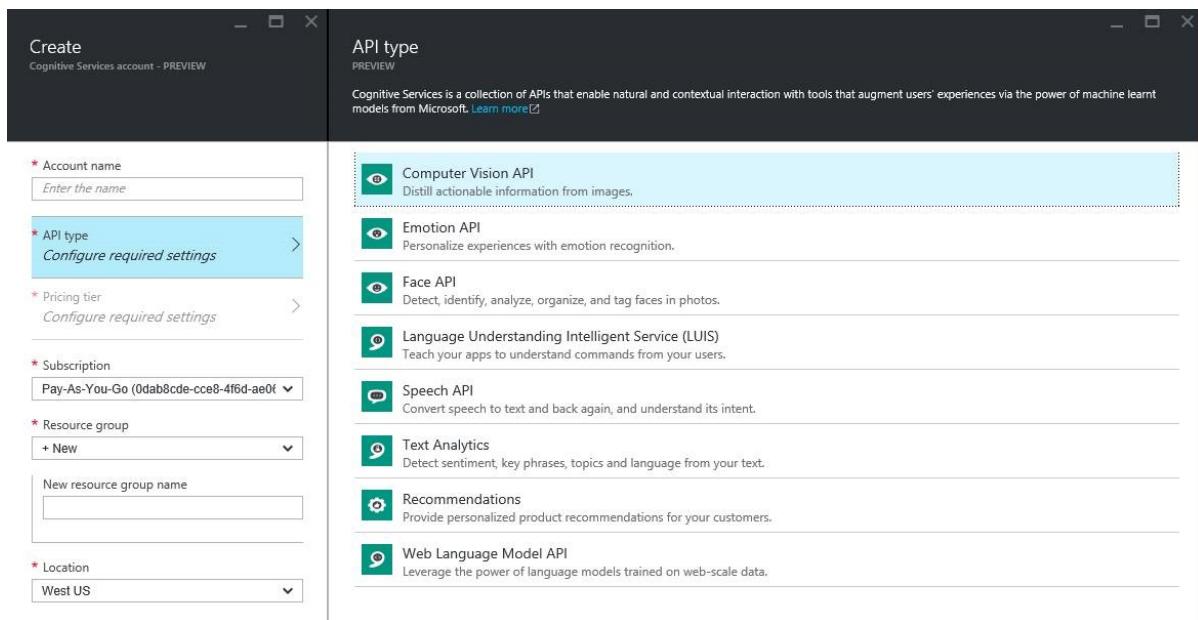
4/12/2017 • 1 min to read • [Edit Online](#)

For unlimited traffic to your HTTP endpoint, you must create a metered key for your account on LUIS. Metered keys provide unlimited traffic to your endpoint following a payment plan. To create your key, follow these steps:

1. Sign in to the [Microsoft Azure Portal](#)
2. Click the green + sign in the upper left-hand panel and search for "Cognitive Services" in the marketplace, then click on **Cognitive Services APIs** and follow the **create experience** to create an API account you are interested in.



3. Choose the API you are interested in by clicking on the **API Type** to create a subscription. In this case, choose the **Language Understanding Intelligent Service (LUIS)** option. Configure the subscription with settings including account name, pricing tiers, etc.



4. Once you have created the API account, you can view the keys generated in the **Settings->Keys** blade. These can be tested in your existing application or by following the LUIS documentation to create a new application.

The screenshot shows three windows side-by-side:

- Left Window (Settings):** Displays the "emotions0-hzf31801" Cognitive Services account - PREVIEW. It shows the "Essentials" section with details like Resource group (hzf), Status (Active), Location (West US), Subscription name (Pay-As-You-Go), and Subscription ID (0dab8cde-cce8-4f6d-ae06-cd6fd134be8b). It also lists API type (Emotion), Pricing tier (Standard), and Endpoint (https://oxfordibiza.azure-api.net/emotion/...).
- Middle Window (Settings):** Shows the "Settings" PREVIEW page with sections for SUPPORT + TROUBLESHOOTING (Audit logs), MANAGE (Properties, Keys, Pricing tier), HELP (Quick start), and RESOURCE MANAGEMENT (Users, Tags). A "Filter settings" search bar is at the top.
- Right Window (Manage keys):** Shows the "Manage keys" interface for the account "emotions0-hzf31801 - PREVIEW". It lists two keys:
  - KEY 1:** Value 013d244fdccf4bd89e78060c11d223aa, with regenerate and copy buttons.
  - KEY 2:** Value 2d175d79ac2e4a8ebb0808d6718404cf, with regenerate and copy buttons.

When you have created your key, you can add it to your account via the application settings/configuration dialog, found in any application.

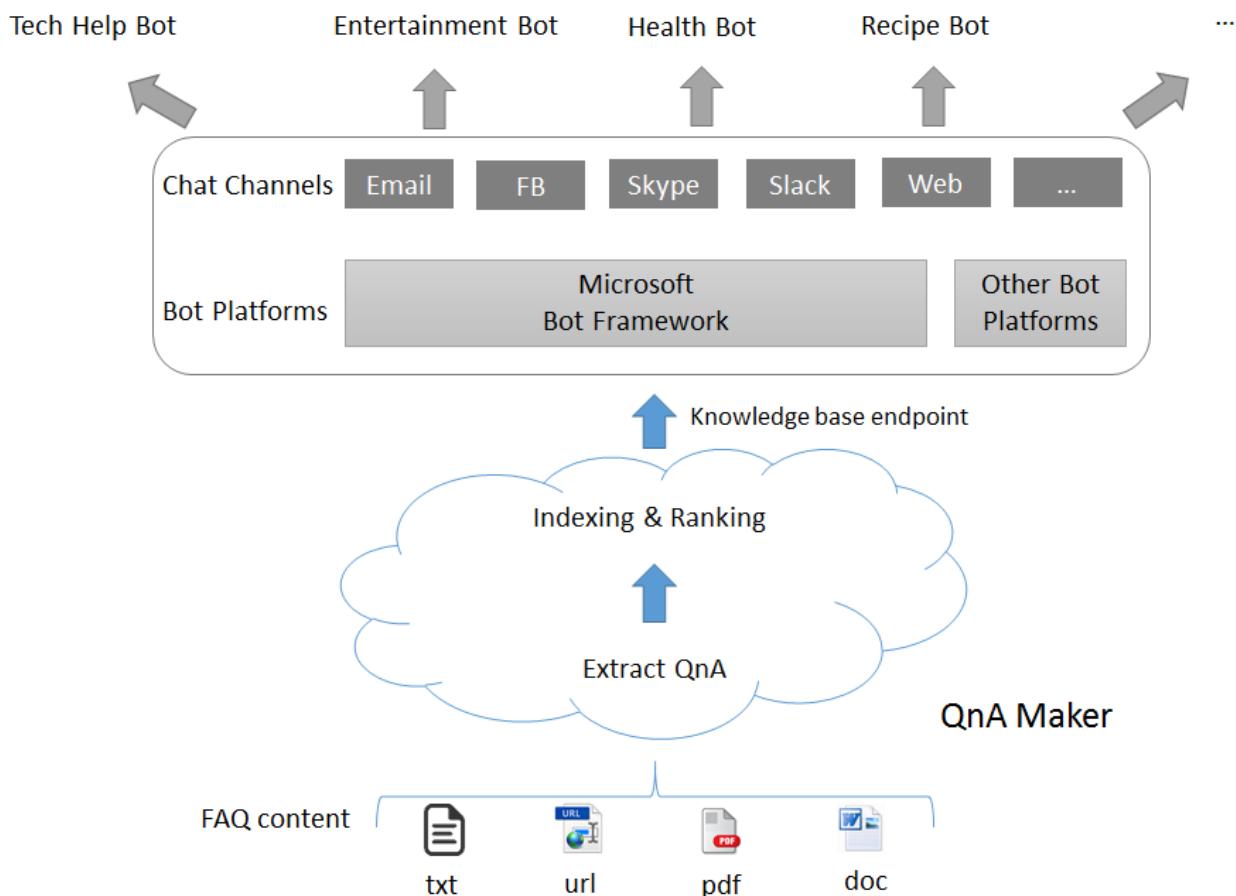
# QnA Maker Overview

4/12/2017 • 1 min to read • [Edit Online](#)

One of the basic requirements in writing your own Bot service is to seed it with questions and answers. In many cases, the questions and answers already exist in content like FAQ URLs/documents, etc.

Microsoft QnA Maker is a free, easy-to-use, REST API and web-based service that trains AI to respond to user's questions in a more natural, conversational way. Compatible across development platforms, hosting services, and channels, QnA Maker is the only question and answer service with a graphical user interface—meaning you don't need to be a developer to train, manage, and use it for a wide range of solutions.

With optimized machine learning logic and the ability to integrate industry-leading language processing with ease, QnA Maker distills masses of information into distinct, helpful answers.



# Quick start video

4/12/2017 • 1 min to read • [Edit Online](#)

This 8-minute video gives you a quick overview of all the features of the QnA Maker tool, and walks you through your first knowledge base creation.

Click [here](#) to see the video on the QnA Maker documentation page.

# Create your knowledge base

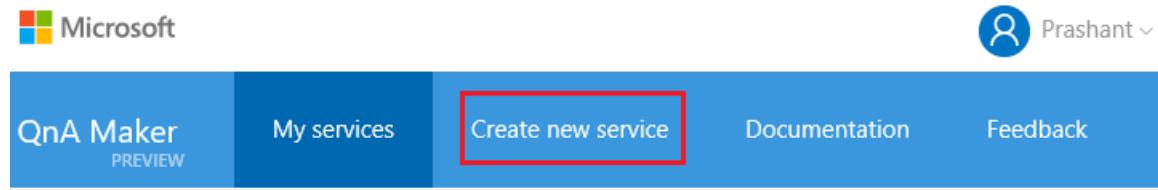
4/12/2017 • 1 min to read • [Edit Online](#)

Creating your knowledge base is as simple as pointing the tool to the existing content, and ingesting the QnA content.

Currently the tool can auto-extract question and answer pairs from most FAQ URLs and documents. If we are not able auto-extract, there is an option to editorially add QnA pairs later.

Follow the below steps to create a new KB.

## Step 1 - Click on Create new service



## Step 2 - Add sources for your KB

# Creating a QnA service

Add sources which contain question and answer pairs you would like to include in your knowledge base.

SERVICE NAME

## What would you like to name your service?

The service name is for your reference and you can change it at anytime.

Name your service

FAQ URL(S)

## What is the URL of your company FAQ page?

This will help us gather relevant data about your business and extract QnA pairs that you can later use in your bot. Here is an [example](#) of a page that would work.

http://

+ Add another

FAQ FILES

## No FAQ URL? No worries. Upload files containing your question and answer pairs.

Supported formats are .tsv, .pdf, .doc, .docx, each under 2MB. Upload up to five files containing questions and answers in sequence. [See an example](#).

Select file...

## Step 3 - Click on Create

**Up next: Crawling your content and creating knowledge base for your service.**

Next the tool will look through your links and documents and create a knowledge base for your service. This will be the structure and "brain" for your new knowledge base service. You'll be able to correct and add to this information in the following step.

Cancel

Create

# Publish your knowledge base

4/12/2017 • 1 min to read • [Edit Online](#)

Once satisfied with the content and relevance of your knowledge base, you can proceed to publishing it as a service. Click on the Publish button.

## Windows download FAQ

[Download Knowledge Base](#) | [Replace Knowledge Base](#)

 Save and retrain

 Publish

*Retrained 4 minutes ago*

## Step 1 - Preview

Before the final publish, you can preview the changes that will affect the knowledge base on final publish. Download the diff file to see what changes will be published.

### Windows download FAQ

#### Review your changes

Source	QnA in production	QnA in current draft	QnA added	QnA deleted
<a href="https://www.microsoft.com/en-us/software-download/faq">https://www.microsoft.com/en-us/software-download/faq</a>	26	26	0	0
Editorial	6	6	0	0
upload - Copy.tsv	0	29	29	0

[Download Diff File](#)

[Cancel](#)

[Publish](#)

## Step 2 - Publish

Once satisfied with the preview, click on Publish.

# Success! Your service has been deployed. What's next?

You can always find the deployment details in your service's settings.

Use the below HTTP request to build your bot. [Learn how.](#)

Sample HTTP request

```
POST /knowledgebases/321595eaaf0347d08f5856df0adb006e/generateAnswer
Host: https://westus.api.cognitive.microsoft.com/qnamaker/v1.0
Ocp-Apim-Subscription-Key: 5f7a25da02df45b5a3045dc155888898
Content-Type: application/json
>{"question":"hi"}
```

Need to fine-tune and refine? Go back and keep editing your service.

[Edit Service](#)

# Test and train

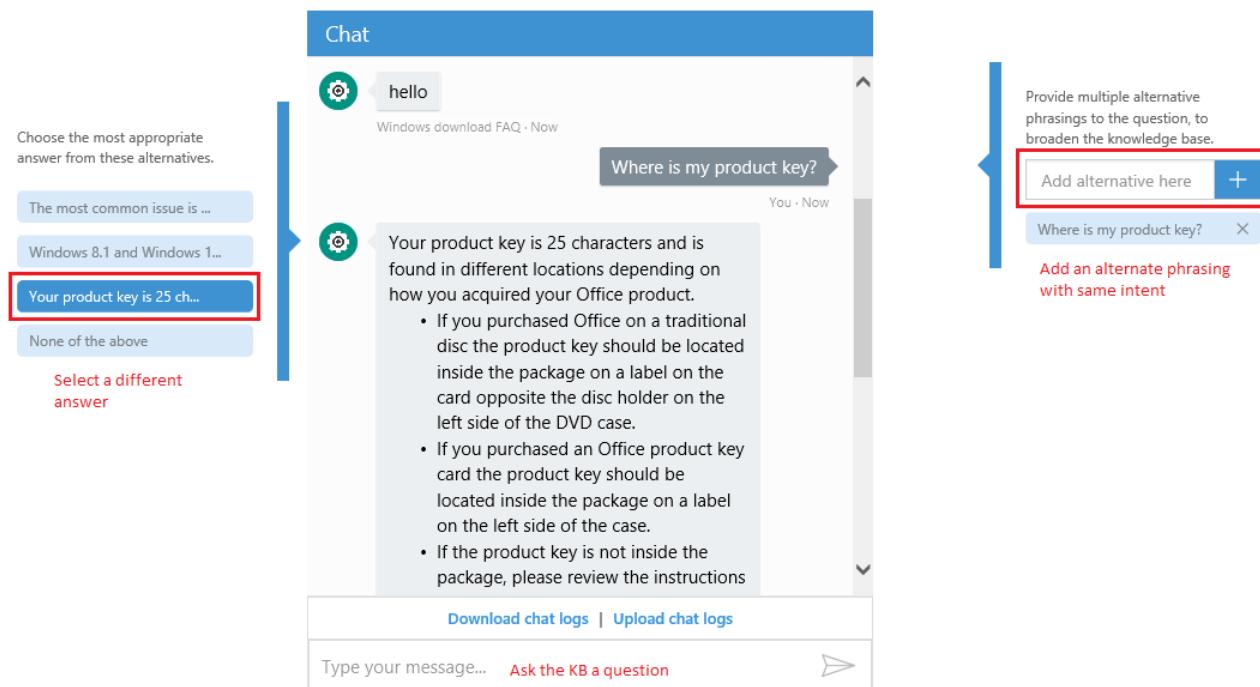
4/12/2017 • 1 min to read • [Edit Online](#)

The relevance of the responses is the most important part of your QnA service. The train feature lets you evaluate the correctness of the responses and correct them and re-train the knowledge base.

There are two ways you can improve the relevance of the responses.

## Chat with your KB

Chat with your knowledge base, to see the relevance of the responses. You can add a variation to an existing question as well as choose a different answer for a question



To bulk upload questions (so you don't need to type every time), click on Upload chat logs. It expects one question per line.

Once uploaded, you can play the questions in order by clicking Show next question.

### WARNING

Only one file can be uploaded at a time. Uploading multiple file will overwrite the previous one.

The screenshot shows the AI Chat interface. At the top, a blue bar says "Chat". Below it is a circular profile icon with a gear and a question mark. A message from "Windows download FAQ" says "Hi! I'm Windows download FAQ. Say 'hi' if you'd like to chat." A timestamp "Windows download FAQ · 1 min ago" is shown. On the left, a sidebar says "Choose the most appropriate answer from these alternatives." with options "hello" and "None of the above". In the main area, a message from "You" says "hi" with timestamp "You · 1 min ago". Below the messages is a toolbar with "Download chat logs" (highlighted with a red box), "Upload chat logs" (highlighted with a red box), and "Bulk upload chatlogs". A text input field "Type your message..." has a paper airplane icon. To the right, a sidebar says "Provide multiple alternative phrasings to the question, to broaden the knowledge base." with a "Add alternative here" button and a "hi" entry. A blue vertical bar connects the sidebar to the main message area.

Make sure you press Save and retrain, to reflect any changes/inputs you have provided.



## Replay live chat logs

A very useful feature is to see what responses the service returns for live traffic, and then train it appropriately.

You can download the live chat traffic hitting your published end-point by clicking on Download chat logs. This downloads all the questions hitting your end-point in descending order of frequency.

Looking at the chat logs, you can decide which questions you want to test and train your knowledge base on, as described in the above section.

This screenshot is similar to the previous one but with different highlighted buttons. The "Download chat logs" button is highlighted with a red box, while the "Upload chat logs" button is not. The rest of the interface is identical to the first screenshot, including the sidebar, messages, toolbar, and sidebar sidebar.

# Update your knowledge base

4/12/2017 • 1 min to read • [Edit Online](#)

There are several ways you might want to update your knowledge base. Choose what is appropriate from the below methods.

## Editorial QnA updates

Edit the QnA table directly. In addition, you can add/delete a QnA Pair, or add an alteration of an existing QnA pair. This is ideal for quick editorial fixes to your knowledge base

KNOWLEDGE BASE | 91 QnA pairs

	Question	Answer
1	hi	Howdy
2	How are you?	I am fine

+ Add new QnA pair

Original source: Editorial

Add alternative phrasing

Delete QnA pair

To save your changes, press Save and retrain button.



## Update the sources

If you need to update the sources of data of your knowledge base, go to the settings tab and update the sources.

## SETTINGS

Service name

Windows download FAQ

### URLs

<https://www.microsoft.com/en-us/software-download/faq>



http://

+ Add another

### Files

upload - Copy.tsv



Select file...

Save and retrain

Once done with changes, click on Save and retrain

## Download and upload

There is also a way to replace your entire knowledge base at one go. This is ideal for bulk updates to your knowledge base. Upload KB expects file format of tab separated columns of Question, Answer and Source.

You have an option to download the entire knowledge base by clicking on Download Knowledge Base, make changes, and then upload the knowledge base.

### WARNING

Uploading a knowledge base overwrites the existing QnAs in the knowledge base.

## Windows download FAQ

[Download Knowledge Base](#) | [Replace Knowledge Base](#)

Save and retrain

Publish

Retrained 4 minutes ago

To save your changes, press Save and retrain button.

Save and retrain

Publish

Retrained 12 minutes ago

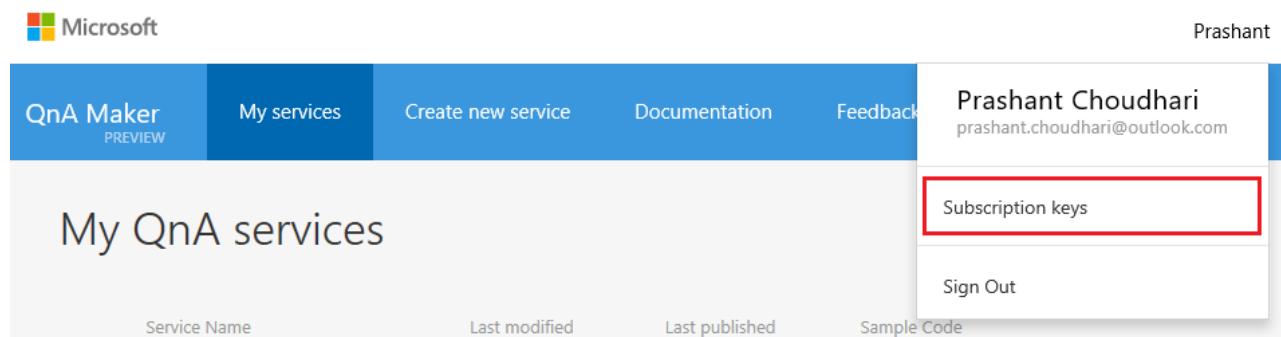
# Authentication & Subscription keys

4/12/2017 • 1 min to read • [Edit Online](#)

You will need a [Microsoft account](#) if you don't already have one, to sign in to the portal.

You will receive a unique pair of keys. The second one is just a spare. Please do not share the secret keys with anyone.

These subscription keys are used to track your usage of the service and need to be part of every request, as mentioned in the API section. To view your subscription keys, go to Settings.



The screenshot shows the Microsoft QnA Maker settings interface. At the top, there's a navigation bar with links for 'QnA Maker PREVIEW', 'My services', 'Create new service', 'Documentation', and 'Feedback'. On the right, a user profile is shown with the name 'Prashant Choudhari' and email 'prashant.choudhari@outlook.com'. A red box highlights the 'Subscription keys' link under the user profile. Below the navigation bar, the main area is titled 'My QnA services'. It has a table with columns for 'Service Name', 'Last modified', 'Last published', and 'Sample Code'. The 'Subscription keys' link is located in the top right corner of this table area.

Here you can view and also refresh your subscription keys, if you suspect they have been compromised.

## Subscription keys

QnA Maker is provided under Cognitive Services Terms.

This free preview provides up to 10 transactions per minute, up to 10,000 transactions per month.

Primary key

.....

[Show](#) | [Copy](#)

Secondary key

.....

[Show](#) | [Copy](#)

Since currently the QnA Maker is a free to use tool, we have the following restrictions of usage per subscription key: **10,000 transactions per month, 10 per minute.** Beyond this your requests will be throttled.

# API V2.0 Reference

4/12/2017 • 1 min to read • [Edit Online](#)

The new [V2.0 APIs](#) let you programmatically manage your knowledge base. Now you can do the following with the APIs

- Create knowledge base
- Delete knowledge base
- Update knowledge base
- Download knowledge base
- Publish knowledge base

Refer [here](#) for detailed documentation.

# API V1.0 Reference

Each published QnA Maker service is exposed as an HTTP endpoint that will take in a question and respond back with the best matched answer and a confidence score. You need the following two GUIDs to access your knowledge base via the HTTP endpoint

- Knowledge Base ID: This is auto-generated by the system for every published KB.
- Subscription Key: These are assigned per account, and is used for metering. See the subscription keys section for more details.

## Sample Request

### HTTP

```
POST /knowledgebases/<Your KB ID>/generateAnswer HTTP/1.1
Host: https://westus.api.cognitive.microsoft.com/qnamaker/v1.0
Ocp-Apim-Subscription-Key: <Your Subscription key>
Content-Type: application/json
Cache-Control: no-cache
{"question": "Question goes here"}
```

### C#

```

string responseString = string.Empty;

var query = "hi"; //User Query
var knowledgebaseId = "YOUR KNOWLEDGE BASE ID"; // Use knowledge base id created.
var qnamakerSubscriptionKey = "YOUR SUBSCRIPTION KEY"; //Use subscription key assigned to you.

//Build the URI
Uri qnamakerUriBase = new Uri("https://westus.api.cognitive.microsoft.com/qnamaker/v1.0");
var builder = new UriBuilder($"{qnamakerUriBase}/knowledgebases/{knowledgebaseId}/generateAnswer");

//Add the question as part of the body
var postBody = $"{{\"question\": \"{query}\", \"knowledgebaseId\": \"{knowledgebaseId}\", \"subscriptionKey\": \"{qnamakerSubscriptionKey}\", \"topScoringOnly\": true}}";

//Send the POST request
using (WebClient client = new WebClient())
{
    //Set the encoding to UTF8
    client.Encoding = System.Text.Encoding.UTF8;

    //Add the subscription key header
    client.Headers.Add("Ocp-Apim-Subscription-Key", qnamakerSubscriptionKey);
    client.Headers.Add("Content-Type", "application/json");
    responseString = client.UploadString(builder.Uri, postBody);
}

```

## Sample Response

The response to the above request will be a JSON with the answer and the confidence score (0-100).

### JSON

```
{ "Answer": "Sample response", "Score": "0" }
```

### C#

```

using Newtonsoft.Json;

private class QnAMakerResult
{
    /// <summary>
    /// The top answer found in the QnA Service.
    /// </summary>
    [JsonProperty(PropertyName = "answer")]
    public string Answer { get; set; }

    /// <summary>
    /// The score in range [0, 100] corresponding to the top answer found in the QnA Service.
    /// </summary>
    [JsonProperty(PropertyName = "score")]
    public double Score { get; set; }
}

//De-serialize the response
QnAMakerResult response;
try
{
    response = JsonConvert.DeserializeObject< QnAMakerResult >(responseString);
}
catch
{
    throw new Exception("Unable to deserialize QnA Maker response string.");
}

```

# FAQs

4/12/2017 • 4 min to read • [Edit Online](#)

## Who are the target audience for the QnA Maker tool?

QnA Maker is primarily meant to provide a FAQ data source which you can query from your Bot/Application. Although developers will find this useful, content owners will especially benefit from this tool. QnA Maker is a completely no-code way of managing the content that powers your Bot/Application.

## How do I login to the QnA Maker Portal?

You can login with your [Microsoft account](#).

## Is the QnA Maker Service free?

Yes, currently the QnA Maker tool is free to use. However, we do meter the usage per account. See the Subscription Keys section of the documentation for details.

## My URLs have valid FAQ content, but the tool cannot extract them. Why not?

It's possible that the tool is not able to auto-extract QnA from valid FAQ URLs. In such cases, you have an option to copy-paste the QnA content in a txt and try ingesting it. Alternately, you can always editorially add content to your knowledge base.

## What format does the tool expect the file content to be?

We support two formats of files for ingestion.

1. .tsv: QnA contained in the format Question<Tab>Answer
2. .txt, .docx, .pdf: QnA contained as regular FAQ content, i.e. sequence of questions and answers.

## How large a knowledge base can I create?

Currently we have a limit of a 20MB knowledge base.

## The updates I made to my knowledge base are not reflected on publish. Why not?

Every edit operation, whether in Table update, Test or Settings needs to be saved before it can be published. Make sure you press the Save and retrain button after every edit operation.

## What is the roadmap of the QnA Maker?

Currently the QnA Maker tool handles semi-structured FAQ content. Eventually the vision is to be able to answer questions from un-structured content as well.

## Do I need to use Bot Framework in order to use QnA Maker?

No, you don't. However, QnA Maker is offered as one of several templates in [Azure Bot Service](#) which enables rapid intelligent bot development powered by Microsoft Bot Framework, and run in a serverless environment. Bots scale based on demand; pay only for the resources you consume.

## Why can't I replace my Knowledge Base with the upload feature?

The format in which upload expects the file is, tab separated columns of Question, Answer and Source.

## When should I refresh my subscription keys?

You should refresh your subscription keys if you suspect that they have been compromised. Any requests with your subscription key will count towards your quota.

## How safe is my knowledge base data?

Every knowledge base content is stored in Azure storage by the QnAMaker tool. You need a combination of knowledge base id and subscription key to access the knowledge base. The knowledge base contents are not used by the tool for any other purpose.

### **Does the KB support rich data?**

The knowledge base supports Markdown. However, the auto-extraction from URLs has limited HTML to Markdown conversion capability. If you want to use full-fledged Markdown, you can modify your content directly in the Table, or upload a KB with the rich content.

### **Does the QnA Maker support non-EN languages?**

The QnA Maker tool ingests and matches data in UTF-16 encoding. This means that any language should work as is. Having said that, we have only extensively tested the relevance of the service for EN yet.

### **Where is the test web-chat URL from the old portal? How do I share my KB with others now?**

In the new Service, we do not have the test URL anymore. The reason being, as part of the cognitive services all calls are being metered. Since the test URL exposes the subscription key and the KB ID, it is a security risk. However, it is still super easy to chat with your KB and share it. Check out the Azure Bot Templates for [Question and Answer Bot](#). You can light up the QnA Bot in Skype in a few clicks, and then share it with anyone.

### **How do I embed the QnA Maker service in my website?**

Follow the below steps to embed the QnA Maker service as a web chat control in your website:

- Create your knowledge base at <https://qnamaker.ai>
- Create your Azure service bot : <https://docs.botframework.com/en-us/azure-bots/build/first-bot>
- Look for the Question and Answer Bot template. Select the KB ID you created in step 1
- Then enable it on web chat channel. Get the embed keys.
- Embed the webchat as shown in <https://docs.botframework.com/en-us/support/embed-chat-control2>

### **What is the format of the downloaded chat logs?**

The chat logs are tab separated files, with the query and the frequency as the columns. Frequency is the number of times the same query was seen. The file is sorted in descending order of frequency. Select questions from the downloaded file you want to test, and then upload it to see what responses the system returned for them.

# Recommendations

4/12/2017 • 1 min to read • [Edit Online](#)

The Recommendations API built with Microsoft Azure Machine Learning helps your customer discover items in your catalog. Customer activity in your digital store is used to recommend items and to improve conversion in your digital store.

The recommendation engine may be trained by uploading data about past customer activity or by collecting data directly from your digital store. When the customer returns to your store you will be able to feature recommended items from your catalog that may increase your conversion rate.

## Resources

[Getting Started Guide](#)

[Collecting Data to Train your Model](#)

[Build Types and Model Quality](#)

[API Reference](#)

## Common Scenarios Supported By Recommendations

### **Frequently Bought Together (FBT) Recommendations**

In this scenario the recommendations engine will recommend items that are likely to be purchased together in the same transaction with a particular item.

For instance, in the example below, customers who bought the Wedge Touch Mouse were also likely to buy the at least one of the following product in the same transaction: Wedge Mobile Keyboard, the Surface VGA Adapter and Surface 2.

### **Item to Item Recommendations**

A common scenario that uses this capability, is "people who visited/clicked on this item, also visited/clicked on this item".

For instance, in the example below, most people who visited the "Wedge Touch Mouse" details page also visited the pages related to other mouse devices.

### **Customer to Item Recommendations**

Given a customer's prior activity, it is possible to recommend items that the customer may be interested in.

For instance, given all movies watched by a customer, it is possible to recommend additional content that may be of interest to the customer.

## Questions?

Feel free to contacts us at [mlapi@microsoft.com](mailto:mlapi@microsoft.com)

# Quick start guide for the Cognitive Services Recommendations API

1/17/2017 • 7 min to read • [Edit Online](#)

This document describes how to onboard your service or application to use the [Recommendations API](#). You can find more details on the Recommendations API and other Cognitive Services [here](#). Throughout this guide, you may also find the [Recommendations API Reference](#) and the [Recommendations UI Quick-Start Guide](#) handy.

## General overview

This document is a step-by-step guide. Our objective is to walk you through the steps necessary to train a model, and to point you to resources that will allow you to consume the model from your production environment.

This exercise will take about 30 minutes.

To use [Recommendations API](#), you need to take the following steps:

1. Create a model - A model is a container of your usage data, catalog data and the recommendation model.
2. Import catalog data - A catalog contains metadata information about the items.
3. Import usage data - Usage data can be uploaded in one of 2 ways (or both):
  - By uploading a file that contains the usage data.
  - Sending data acquisition events. Usually you upload a usage file in order to be able to create an initial recommendation model (bootstrap) and use it until the system gathers enough data by using the data acquisition format.
4. Build a recommendation model - This is an asynchronous operation in which the recommendation system takes all the usage data and creates a recommendation model. This operation can take several minutes or several hours depending on the size of the data and the build configuration parameters. When triggering the build, you will get a build ID. Use it to check when the build process has ended before starting to consume recommendations.
5. Consume recommendations - Get recommendations for a specific item or list of items.

## Let's get started!

You will start building a Recommendations model. Then we'll guide you on how to use the results generated by the model to power recommendations on an e-commerce site.

### Task 1 - Signing up for the Recommendations API

In this task, you'll sign up for the Recommendations API service, and create a recommendations model.

1. Go to the **Azure Portal** at <http://portal.azure.com/> and sign in with your Azure account.
2. Click on **+ New**.
3. Select the **Intelligence** option.
4. Select the **Cognitive Services APIs** product. This product will allow you to start a subscription for any of the cognitive services APIs (Face, Text Analytics, Computer Vision, etc.). Today we will focus on the Recommendations API.
5. On the Cognitive Services API landing page, enter the **Account name** for your Recommendations subscription. (For instance: "MyRecommendations"). This name should not have any spaces in it.
6. On **API type**, select **Recommendations**.
7. On **Pricing tier**, you can select a plan. You may select the **Free** tier for 10,000 transactions/month. This is a free plan, so it is a good way to start trying the system. Once you go to production, we recommend you consider your request volume and change the plan type accordingly. (Note: You can have only one Free tier subscription

- at a time)
8. Select a **Resource Group**, or create a new one if you don't have one already.
  9. You may change other elements in the Create dialog. We should point out that the resource provider today is only supported from United States data centers.
  10. Once you are done with any selections, click **Create**.
  11. Wait a few minutes for the resource to be deployed. Once it is deployed, you can go to the **Keys** section in the **Settings** blade where you will be provided a primary and secondary key to use the API. Copy the primary key, as you'll need it when creating your first model.

#### Task 2 - Did you bring your data?

The Recommendations API will learn from your catalog and your transactions in order to provide good product recommendations. That means you need to feed it with good data about your products (We call this a **catalog** file) and a set of transactions large enough for it to find interesting patterns of consumption (We call this **usage**).

1. Usually you would query your transactions database for these pieces of information. Just in case you don't have them handy, we have provided some sample data for you based on Microsoft Store transaction data.

You can download the data from [here](#). Copy and unpack the MsStoreData.Zip into a folder on your local machine.

#### NOTE

The sample code that you will download and run in Task 3 has sample data already embedded inside it -- so this task is optional. That said, this Task will allow you to download more realistic data sets and allow you to understand the inputs into the Recommendations API better.

2. Now let's take a look at the catalog file. Navigate to the location where you copied the data. Open the catalog file in **notepad**.

You will notice that the catalog file is pretty simple. It has the following format

```
<itemid>,<item name>,<product category>
```

```
AAA-04294,OfficeLangPack 2013 32/64 E34 Online DwnLd,Office  
AAA-04303,OfficeLangPack 2013 32/64 ET Online DwnLd,Office  
C9F-00168,KRUSELL Kiruna Flip Cover for Nokia Lumia 635 - Camel,Accessories
```

We should point out that a catalog file can be much richer, for instance you can add metadata about the products (We call these *item features*). You should see the [catalog format](#) section in the API Reference for more details on the catalog format.

3. Let's do the same with the usage data. You will notice that the usage date is of the format

```
<User Id>,<Item Id>,<Time Stamp>,<Event> .
```

```
00037FFEA61FCA16,288186200,2015/08/04T11:02:52,Purchase  
0003BFFDD4C2148C,297833400,2015/08/04T11:02:50,Purchase  
0003BFFDD4C2118D,297833300,2015/08/04T11:02:40,Purchase  
00030000D16C4237,297833300,2015/08/04T11:02:37,Purchase  
0003BFFDD4C20B63,297833400,2015/08/04T11:02:12,Purchase  
00037FFEC8567FB8,297833400,2015/08/04T11:02:04,Purchase
```

Notice that the first three elements are mandatory. The event type is optional. You can check out the [usage format](#) for more information on this topic.

#### How much data do you need?

Well, it really depends on the usage data itself. The system learns when users buy different items. For some builds like FBT, it is important to know which items are purchased in the same transactions. (We call this *co-occurrences*). A good rule of thumb is to have most items be in 20 transactions or more, so if you had 10,000 items in your catalog, we would recommend that you have at least 20 times that number of transactions or about 200,000 transactions. Once again, this is a rule of thumb. You will need to experiment with your data.

### Task 3 - Creating a recommendations model

Now that you have an account and you have data, let's create your first model.

In this task, you will use the sample application to build your first model.

1. First of all you should be aware of the [Recommendations API Reference](#).
2. Download the [Sample Application](#) to a local folder.
3. Open in Visual Studio the **RecommendationsSample.sln** solution located in the **C#** folder.
4. Open the **SampleApp.cs** file. Note that the following steps in the file:
  - Create a Model
  - Add a catalog file
  - Add a usage file
  - Trigger a build for the Model
  - Get a recommendation based on a pair of items
5. Replace the value for the **AccountKey** field with the key from Task 1.
6. Step through the solution, and you will see how a model gets created.
7. Try replacing the catalog and usage files you just downloaded to create a new model for the Microsoft Store, or for book recommendations. You will need to change the model name as well, and the items for which you request recommendations.
8. When the model is created, take note of the **model ID** as you will need it when requesting recommendations in your production environment.

Learn more about build types and how to evaluate the quality of builds [here](#).

### Putting your model in production!

Now that you understand how to create a model and consume recommendations, the next step is to put it in production on your website, mobile application or integrate into your CRM or ERP system. Obviously, each of these implementations would be different. Since the Recommendations API are requested as a web-service, you should be able to call it from any of these different environments easily.

#### IMPORTANT

If you want to show recommendations from a public client (for instance, your ecommerce site), you should create a proxy server to provide the recommendations. This is important so that your API Key is not exposed to external (potentially untrusted) entities.

Here are a few ideas of locations where you can use Recommendations:

#### Product Page

#### Item Recommendations

If the model was trained on purchase data, it will allow your customer to *discover products that are likely to be of interest to people that have purchased the source item*.

If the model was trained on click data, it will allow your customer to *discover products that are likely to be visited*

*by people that have visited the source item.* This type of model may return similar items.

## Frequently Bought Together Recommendations

A frequently bought together build could be trained, so you can get sets of items are likely to be purchased together with this item.

### Check out Page

## Item Recommendations

A recommendations model could take as input a list of items. So you could pass all the items in the basket as input to get recommendations. In this case, the model will provide recommendations that are of interest given all the items in the basket.

### Landing Page

## User Recommendations

A recommendations model can take as input the user id. This will use the history of transactions by that user to provide personalized recommendations to the user specified.

Check out the [Get Item Recommendations Documentation](#).

### What's next?

Congratulations if you have made it this far! To learn more you can visit the complete [Recommendations API Reference](#) If you have questions, don't hesitate to contact us at [mlapi@microsoft.com](mailto:mlapi@microsoft.com)

# Build types and model quality

1/17/2017 • 10 min to read • [Edit Online](#)

## Supported build types

The Recommendations API currently supports two build types: *recommendation* and *FBT*. Each is built using different algorithms, and each has different strengths. This document describes each of these builds and techniques for comparing the quality of the models generated.

If you have not done so already, we recommend that you complete the [quick start guide](#).

### Recommendation build type

The recommendation build type uses matrix factorization to provide recommendations. It generates [latent feature](#) vectors based on your transactions to describe each item, and then uses those latent vectors to compare items that are similar.

If you train the model based on purchases made in your electronics store and provide a Lumia 650 phone as the input to the model, the model will return a set of items that tend to be purchased by people who are likely to purchase a Lumia 650 phone. The items may not be complementary. In this example, it is possible that the model will return other phones because people who like the Lumia 650 may like other phones.

The recommendation build has two capabilities that make it attractive:

#### The recommendation build supports *cold item placement*

Items that do not have significant usage are called cold items. For instance, if you receive a shipment of a phone you have never sold before, the system cannot infer recommendations for this product on transactions alone. This means that the system should learn from information about the product itself.

If you want to use cold item placement, you need to provide features information for each of your items in the catalog. Following is what the first few lines of your catalog may look like (note the key=value format for the features).

```
6CX-00001, Surface Pro2, Surface,, Type=Hardware, Storage=128 GB, Memory=4G, Manufacturer=Microsoft
```

```
73H-00013, Wake Xbox 360, Gaming,, Type=Software, Language=English, Rating=Mature
```

```
WAH-0F05, Minecraft Xbox 360, Gaming,, * Type=Software, Language=Spanish, Rating>Youth
```

You also need to set the following build parameters:

BUILD PARAMETER	NOTES
<i>useFeaturesInModel</i>	Set to <b>true</b> . Indicates if features can be used to enhance the recommendation model.
<i>allowColdItemPlacement</i>	Set to <b>true</b> . Indicates if the recommendation should also push cold items via feature similarity.
<i>modelingFeatureList</i>	Comma-separated list of feature names to be used in the recommendation build to enhance the recommendation. For instance, "Language,Storage" for the preceding example.

## The recommendation build supports user recommendations

A recommendation build supports [user recommendations](#). This means that it can provide personalized recommendations for users based on their transaction histories. For user recommendations, you might provide the user ID or the recent history of transactions for that user.

One classic example of where you might want to apply user recommendations is at sign-in on the welcome page. There you can promote content that applies to the specific user.

You might also want to apply a recommendations build type when the user is about to check out. At that point, you have the list of items the user is about to purchase, and you can provide recommendations based on the current market basket.

### Recommendations build parameters

NAME	DESCRIPTION	TYPE, VALID VALUES, (DEFAULT VALUE)
<i>NumberOfModelIterations</i>	The number of iterations the model performs is reflected by the overall compute time and the model accuracy. The higher the number, the more accurate the model, but the compute time takes longer.	Integer, 10 to 50 (40)
<i>NumberOfModelDimensions</i>	The number of dimensions relates to the number of features the model will try to find within your data. Increasing the number of dimensions will allow better fine-tuning of the results into smaller clusters. However, too many dimensions will prevent the model from finding correlations between items.	Integer, 10 to 40 (20)
<i>ItemCutOffLowerBound</i>	Defines the minimum number of usage points an item should be in for it to be considered part of the model.	Integer, 2 or more (2)
<i>ItemCutOffUpperBound</i>	Defines the maximum number of usage points an item should be in for it to be considered part of the model.	Integer, 2 or more (2147483647)
<i>UserCutOffLowerBound</i>	Defines the minimum number of transactions a user must have performed to be considered part of the model.	Integer, 2 or more (2)
<i>UserCutOffUpperBound</i>	Defines the maximum number of transactions a user must have performed to be considered part of the model.	Integer, 2 or more (2147483647)
<i>UseFeaturesInModel</i>	Indicates if features can be used to enhance the recommendation model.	Boolean Default: True

NAME	DESCRIPTION	TYPE, VALID VALUES, (DEFAULT VALUE)
<i>ModelingFeatureList</i>	Comma-separated list of feature names to be used in the recommendation build to enhance the recommendation. It depends on the features that are important.	String, up to 512 chars
<i>AllowColdItemPlacement</i>	Indicates if the recommendation should also push cold items via feature similarity.	Boolean Default: False
<i>EnableFeatureCorrelation</i>	Indicates if features can be used in reasoning.	Boolean Default: False
<i>ReasoningFeatureList</i>	Comma-separated list of feature names to be used for reasoning sentences, such as recommendation explanations. It depends on the features that are important to customers.	String, up to 512 chars
<i>EnableU2I</i>	Enable personalized recommendations, also called user to item (U2I) recommendations.	Boolean Default: True
<i>EnableModelingInsights</i>	Defines whether offline evaluation should be performed to gather modeling insights (that is, precision and diversity metrics). If set to true, a subset of the data will not be used for training because it will need to be reserved for testing of the model. Read more about <a href="#">offline evaluations</a> .	Boolean Default: False
<i>SplitterStrategy</i>	If enable modeling insights is set to <i>true</i> , this is how data should be split for evaluation purposes.	String, <i>RandomSplitter</i> or <i>LastEventSplitter</i> Default: RandomSplitter

## FBT build type

The frequently bought together (FBT) build does an analysis that counts the number of times two or three different products co-occur together. It then sorts the sets based on a similarity function (**co-occurrences**, **Jaccard**, **lift**).

Think of **Jaccard** and **lift** as ways to normalize the co-occurrences. This means that the items will be returned only if they were purchased together with the seed item.

In our Lumia 650 phone example, phone X will be returned only if phone X was purchased in the same session as the Lumia 650 phone. Because this may be unlikely, we would expect items complementary to the Lumia 650 to be returned; for instance, a screen protector, or a power adapter for the Lumia 650.

Currently, two items are assumed to be purchased in the same session if they occur in a transaction with the same user ID and timestamp.

FBT builds do not support cold items, because by definition they expect two items to be purchased in the same transaction. While FBT builds can return sets of items (triplets), they do not support personalized recommendations because they accept a single seed item as the input.

## FBT build parameters

NAME	DESCRIPTION	TYPE, VALID VALUES, (DEFAULT VALUE)
<i>FbtSupportThreshold</i>	How conservative the model is. Number of co-occurrences of items to be considered for modeling.	Integer, 3-50 (6)
<i>FbtMaxItemSetSize</i>	Bounds the number of items in a frequent set.	Integer 2-3 (2)
<i>FbtMinimalScore</i>	Minimal score that a frequent set should have to be included in the returned results. The higher the better.	Double 0 and above (0)
<i>FbtSimilarityFunction</i>	Defines the similarity function to be used by the build. <b>Lift</b> favors serendipity, <b>co-occurrence</b> favors predictability, and <b>Jaccard</b> is a compromise between the two.	String, <i>cooccurrence, lift, jaccard</i> Default: <i>jaccard</i>

## Build evaluation and selection

This guidance might help you determine whether you should use a recommendations build or an FBT build, but it does not provide a definitive answer in cases where you could use either of them. Also, even if you know that you want to use an FBT build type, you might still want to choose **Jaccard** or **lift** as the similarity function.

The best way to select between two different builds is to test them in the real world (online evaluation) and track a conversion rate for the different builds. The conversion rate could be measured based on recommendation clicks, the number actual purchases from recommendations shown, or even on the actual purchase amounts when the different recommendations were shown. You may select your conversion rate metric based on your business objective.

In some cases, you may want to evaluate the model offline before you put it in production. While offline evaluation is not a replacement for online evaluation, it can serve as a metric.

## Offline evaluation

The goal of an offline evaluation is to predict precision (the number of users that will purchase one of the recommended items) and the diversity of recommendations (the number of items that are recommended). As part of the precision and diversity metrics evaluation, the system finds a sample of users and splits the transactions for those users into two groups: the training dataset and the test dataset.

### NOTE

To use offline metrics, you must have timestamps in your usage data. Time data is required to split usage correctly between training and test datasets.

Also, offline evaluation may not yield results for small usage files. For the evaluation to be thorough, there should be a minimum of 1,000 usage points in the test dataset.

### Precision-at-k

The following table represents the output of the precision-at-k offline evaluation.

K	1	2	3	4	5
Percentage	13.75	18.04	21	24.31	26.61
Users in test	10,000	10,000	10,000	10,000	10,000
Users considered	10,000	10,000	10,000	10,000	10,000
Users not considered	0	0	0	0	0

## K

In the preceding table,  $k$  represents the number of recommendations shown to the customer. The table reads as follows: "If during the test period, only one recommendation was shown to the customers, only 13.75 of the users would have purchased that recommendation." This statement is based on the assumption that the model was trained with purchase data. Another way to say this is that the precision at 1 is 13.75.

You will notice that as more items are shown to the customer, the likelihood of the customer purchasing a recommended item goes up. For the preceding experiment, the probability almost doubles to 26.61 percent when 5 items are recommended.

### Percentage

The percentage of users that interacted with at least one of the  $k$  recommendations is shown. The percentage is calculated by dividing the number of users that interacted with at least one recommendation by the total number of users considered. See Users considered for more information.

### Users in test

Data in this row represents the total number of users in the test dataset.

### Users considered

A user is only considered if the system recommended at least  $k$  items based on the model generated using the training dataset.

### Users not considered

Data in this row represents any users not considered. The users that did not receive at least  $k$  recommended items.

User not considered = users in test – users considered

## Diversity

Diversity metrics measure the type of items recommended. The following table represents the output of the diversity offline evaluation.

PERCENTILE BUCKET	0-90	90-99	99-100
Percentage	34.258	55.127	10.615

Total items recommended: 100,000

Unique items recommended: 954

### Percentile buckets

Each percentile bucket is represented by a span (minimum and maximum values that range between 0 and 100). The items close to 100 are the most popular items, and the items close to 0 are the least popular. For instance, if the percentage value for the 99-100 percentile bucket is 10.6, it means that 10.6 percent of the recommendations returned only the top one percent most popular items. The percentile bucket minimum value is inclusive, and the maximum value is exclusive, except for 100.

### **Unique items recommended**

The unique items recommended metric shows the number of distinct items that were returned for evaluation.

### **Total items recommended**

The total items recommended metric shows the number of items recommended. Some may be duplicates.

### **Offline evaluation metrics**

The precision and diversity offline metrics may be useful when you select which build to use. At build time, as part of the respective FBT or recommendation build parameters:

- Set the *enableModelingInsights* build parameter to **true**.
- Optionally, select the *splitterStrategy* (Either *RandomSplitter* or *LastEventSplitter*). *RandomSplitter* splits the usage data in train and test sets based on the given *randomSplitterParameters* test percent and random seed values. *LastEventSplitter* splits the usage data in train and test sets based on the last transaction for each user.

This will trigger a build that uses only a subset of the data for training and uses the rest of the data to compute evaluation metrics. After the build is completed, to get the output of the evaluation, you need to call the [Get build metrics API](#), passing the respective *modelId* and *buildId*.

Following is the JSON output for the sample evaluation.

```
{
  "Result": {
    "precisionItemRecommend": null,
    "precisionUserRecommend": {
      "precisionMetrics": [
        {
          "k": 1,
          "percentage": 13.75,
          "usersInTest": 10000,
          "usersConsidered": 10000,
          "usersNotConsidered": 0
        },
        {
          "k": 2,
          "percentage": 18.04,
          "usersInTest": 10000,
          "usersConsidered": 10000,
          "usersNotConsidered": 0
        },
        {
          "k": 3,
          "percentage": 21.0,
          "usersInTest": 10000,
          "usersConsidered": 10000,
          "usersNotConsidered": 0
        },
        {
          "k": 4,
          "percentage": 24.31,
          "usersInTest": 10000,
          "usersConsidered": 10000,
          "usersNotConsidered": 0
        },
        {
          "k": 5,
          "percentage": 26.61,
          "usersInTest": 10000,
          "usersConsidered": 10000,
          "usersNotConsidered": 0
        }
      ],
      "error": null
    }
  }
}
```

```
"diversityItemRecommendation": null,
"diversityUserRecommend": {
  "percentileBuckets": [
    {
      "min": 0,
      "max": 90,
      "percentage": 34.258
    },
    {
      "min": 90,
      "max": 99,
      "percentage": 55.127
    },
    {
      "min": 99,
      "max": 100,
      "percentage": 10.615
    }
  ],
  "totalItemsRecommended": 100000,
  "uniqueItemsRecommended": 954,
  "uniqueItemsInTrainSet": null,
  "error": null
},
{
  "Id": 1,
  "Exception": null,
  "Status": 5,
  "IsCanceled": false,
  "IsCompleted": true,
  "CreationOptions": 0,
  "AsyncState": null,
  "IsFaulted": false
}
```

# Collecting Data to Train your Model

1/17/2017 • 6 min to read • [Edit Online](#)

The Recommendations API learns from your past transactions to find what items should be recommended to a particular user.

After you have created a model, you will need to provide two pieces of information before you can do any training: a catalog file, and usage data.

If you have not done so already, we encourage you to complete the [quick start guide](#).

## Catalog Data

### Catalog file format

The catalog file contains information about the items you are offering to your customer. The catalog data should follow the following format:

- Without features - <Item Id>,<Item Name>,<Item Category>[,<Description>]
- With features - <Item Id>,<Item Name>,<Item Category>,[<Description>],<Features list>

### Sample Rows in a Catalog File

Without features:

```
AAA04294,Office Language Pack Online DwnLd,Office
AAA04303,Minecraft Download Game,Games
C9F00168,Kiruna Flip Cover,Accessories
```

With features:

```
AAA04294,Office Language Pack Online DwnLd,Office,, softwaretype=productivity, compatibility=Windows
BAB04303,Minecraft DwnLd,Games, softwaretype=gaming,, compatibility=iOS, agegroup=all
C9F00168,Kiruna Flip Cover,Accessories, compatibility=lumia,, hardwaretype=mobile
```

### Format details

NAME	MANDATORY	TYPE	DESCRIPTION
Item Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 50	Unique identifier of an item.
Item Name	Yes	Any alphanumeric characters Max length: 255	Item name.
Item Category	Yes	Any alphanumeric characters Max length: 255	Category to which this item belongs (e.g. Cooking Books, Drama...); can be empty.
Description	No, unless features are present (but can be empty)	Any alphanumeric characters Max length: 4000	Description of this item.

NAME	MANDATORY	TYPE	DESCRIPTION
Features list	No	Any alphanumeric characters Max length: 4000; Max number of features:20	Comma-separated list of feature name=feature value that can be used to enhance model recommendation.

#### Uploading a Catalog file

Look at the [API reference](#) for uploading a catalog file.

Note that the content of the catalog file should be passed as the request body.

If you upload several catalog files to the same model with several calls, we will insert only the new catalog items. Existing items will remain with the original values. You cannot update catalog data by using this method.

Note: The maximum file size is 200MB. The maximum number of items in the catalog supported is 100,000 items.

## Why add features to the catalog?

The recommendations engine creates a statistical model that tells you what items are likely to be liked or purchased by a customer. When you have a new product that has never been interacted with it is not possible to create a model on co-occurrences alone. Let's say you start offering a new "children's violin" in your store, since you have never sold that violin before you cannot tell what other items to recommend with that violin.

That said, if the engine knows information about that violin (i.e. It's a musical instrument, it is for children ages 7-10, it is not an expensive violin, etc.), then the engine can learn from other products with similar features. For instance, you have sold violin's in the past and usually people that buy violins tend to buy classical music CDs and sheet music stands. The system can find these connections between the features and provide recommendations based on the features while your new violin has little usage.

Features are imported as part of the catalog data, and then their rank (or the importance of the feature in the model) is associated when a rank build is done. Feature rank can change according to the pattern of usage data and type of items. But for consistent usage/items, the rank should have only small fluctuations. The rank of features is a non-negative number. The number 0 means that the feature was not ranked (happens if you invoke this API prior to the completion of the first rank build). The date at which the rank was attributed is called the score freshness.

#### Features are Categorical

This means that you should create features that resemble a category. For instance, price=9.34 is not a categorical feature. On the other hand, a feature like priceRange=Under5Dollars is a categorical feature. Another common mistake is to use the name of the item as a feature. This would cause the name of an item to be unique so it would not describe a category. Make sure the features represent categories of items.

#### How many/which features should I use?

Ultimately the Recommendations build supports building a model with up to 20 features. You could assign more than 20 features to the items in your catalog, but you are expected to do a ranking build and pick only the features that rank high. (A feature with a rank of 2.0 or more is a really good feature to use!).

#### When are features actually used?

Features are used by the model when there is not enough transaction data to provide recommendations on transaction information alone. So features will have the greatest impact on "cold items" – items with few transactions. If all your items have sufficient transaction information you may not need to enrich your model with features.

#### Using product features

To use features as part of your build you need to:

1. Make sure your catalog has features when you upload it.
2. Trigger a ranking build. This will do the analysis on the importance/rank of the features.
3. Trigger a recommendations build, setting the following build parameters: Set useFeaturesInModel to true, allowColdItemPlacement to true, and modelingFeatureList should be set to the comma separated list of features that you want to use to enhance your model. See [Recommendations build type parameters](#) for more information.

## Usage Data

A usage file contains information about how those items are used, or the transactions from your business.

### Usage Format details

A usage file is a CSV (comma separated value) file where each row in a usage file represents an interaction between a user and an item. Each row is formatted as follows:

```
<User Id>,<Item Id>,<Time>,[<Event>]
```

NAME	MANDATORY	TYPE	DESCRIPTION
User Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 255	Unique identifier of a user.
Item Id	Yes	[A-z], [a-z], [0-9], [_] (Underscore), [-] (Dash) Max length: 50	Unique identifier of an item.
Time	Yes	Date in format: YYYY/MM/DDTHH:MM:SS (e.g. 2013/06/20T10:00:00)	Time of data.
Event	No	One of the following: <ul style="list-style-type: none"><li>• Click</li><li>• RecommendationClick</li><li>• AddShopCart</li><li>• RemoveShopCart</li><li>• Purchase</li></ul>	The type of transaction.

### Sample Rows in a Usage File

```
00037FFEA61FCA16,288186200,2015/08/04T11:02:52,Purchase
0003BFFDD4C2148C,297833400,2015/08/04T11:02:50,Purchase
0003BFFDD4C2118D,297833300,2015/08/04T11:02:40,Purchase
00030000D16C4237,297833300,2015/08/04T11:02:37,Purchase
0003BFFDD4C20B63,297833400,2015/08/04T11:02:12,Purchase
00037FFEC8567FB8,297833400,2015/08/04T11:02:04,Purchase
```

### Uploading a usage file

Look at the [API reference](#) for uploading usage files. Note that you need to pass the content of the usage file as the body of the HTTP call.

Note:

Maximum file size: 200MB. You may upload several usage files.

You need to upload a catalog file before you start adding usage data to your model. Only items in the catalog file will be used during the training phase. All other items will be ignored.

## How much data do you need?

The quality of your model is heavily dependent on the quality and quantity of your data. The system learns when users buy different items (We call this co-occurrences). For FBT builds, it is also important to know which items are purchased in the same transactions.

A good rule of thumb is to have most items be in 20 transactions or more, so if you had 10,000 items in your catalog, we would recommend that you have at least 20 times that number of transactions or about 200,000 transactions. Once again, this is a rule of thumb. You will need to experiment with your data.

Once you have created a model, you can perform an [offline evaluation](#) to check how well your model is likely to perform.

# Building a model with the Recommendations UI

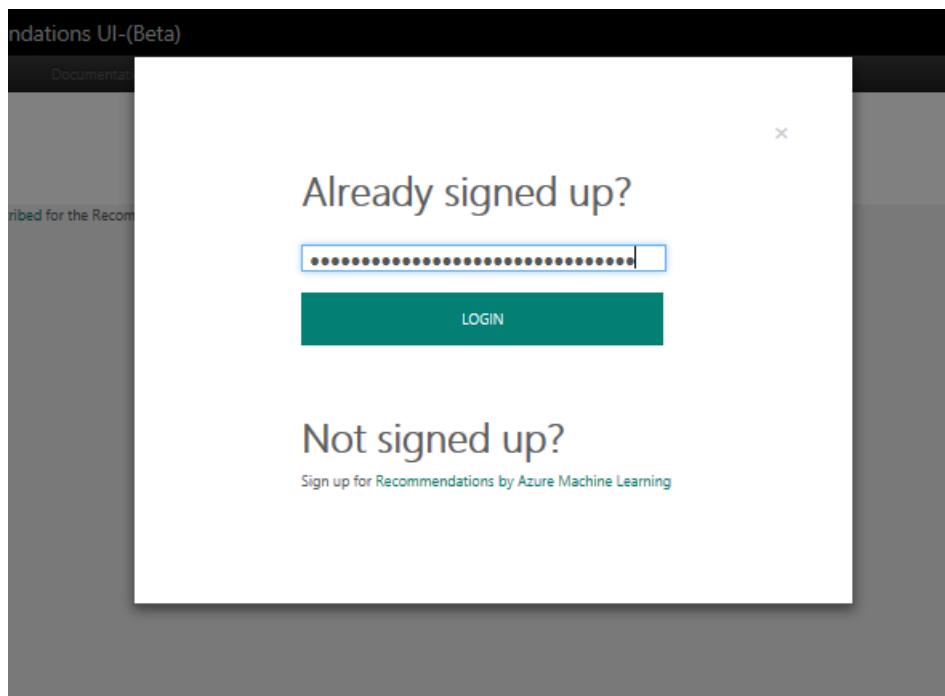
1/17/2017 • 4 min to read • [Edit Online](#)

This document is a step-by-step guide. Our objective is to walk you through the steps necessary to train a model using the [Recommendations UI](#). Going through the exercise allows you to understand the process for building a model before you do it programmatically. It also familiarizes you with the UI, which is handy as you start using the service.

This exercise takes about 30 minutes.

## Step 1 - Sign in to the Recommendations UI

1. If you have not done so already, you need to [sign-up](#) for a new [Recommendations API](#) subscription. You can sign up for the service on **Azure** at <http://portal.azure.com/> and sign in with your Azure account. Detailed instructions on the sign up process are provided in *Task 1* of the [Getting Started Guide](#)
2. Once you have obtained a **Key** for your Recommendations API subscription, go to [Recommendations UI](#).
3. Log in to the portal by entering your key in the **Account Key** field, and then click the **Login** button.



## Step 2 - Let's gather some training data

Before you can create a build, the engine needs two pieces of information: a catalog file and a set of usage files.

The catalog file contains information about the items you are offering to your customer. A usage file contains information about how those items are used, or the transactions from your business.

Usually you would query your store database for these pieces of information. Today, we have provided some sample data for you so that you can learn how to use the Recommendations API.

You can download the data from <http://aka.ms/RecoSampleData>. Copy and unpack the **Books.Zip** file into a folder on your local machine. For instance, **c:\data**.

Detailed information on the schema of the catalog and usage files can be found in the [Collecting Data To Train your Model](#) article.

For this exercise, we are going to work with a small file so that you don't have very to wait too long for training. If you want to try a more realistic file, we have also placed **MsStoreData.zip** that contains sample transactions from the Microsoft Store in the [same location](#).

## Step 3 - Create a project and upload catalog and usage data

Upon logging in to the [Recommendations UI](#), you see the Projects Page. If you have previously created any projects, you should see them here. A project (also known as *a model* in the API reference) is a container for your catalog and usage data. You can create several *builds* inside the project. We will walk you through the process in the next steps.

1. To create a new project, type the name on the text box (Something like "MyFirstModel" would work) and click **Add Project**.

Name	Model ID	Created
MSStore2	06d5e608-882e-4763-9110-9924c09369ba	5/21/2016 12:05:46 PM
MyNewModel	3f3f1120-747e-4187-9f16-0ed937aaa460	9/22/2016 5:21:18 PM
MyModel3	95cdc27f-c203-4539-8852-c43eacd2119b	5/24/2016 10:39:45 AM
Shoes	a56c5702-1a07-4044-b7f3-48ded9e44a7c	8/27/2016 11:30:14 AM

2. Once the project gets created, click the **Browse for File** button on the **Add a Catalog File** section. Upload the catalog you got in step 2. If you saved it at *c:\data*, you need to navigate to that folder.

MyFirstModel Model ID: 89f8a410-87a3-4b9d-8c19-18acbc917210

Step 1: Add a catalog file (**schema**)

BROWSE FOR FILE...

Step 2: Add usage files (**schema**)

Step 3: Create a **build**

NEW BUILD

3. After the catalog is uploaded, click the **Browse for File** button on the **Add Usage Files** section. Add the *usage\_large.txt* file.

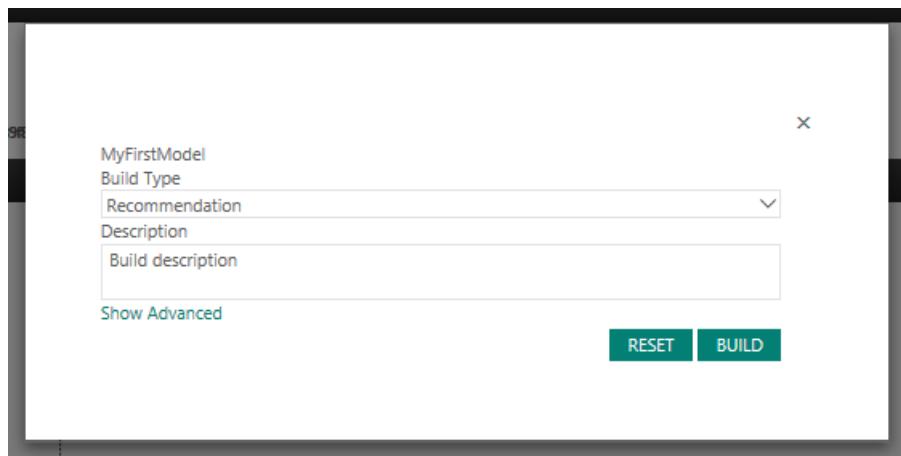
**What if I have a large file of usage data?**

Only usage files smaller than 200-MB can be uploaded. That said, the system can hold up to 2-GB worth of transaction data, so you can upload more than one file if necessary. You may not need that much data to generate a good model, it's not just the size of the data that matters, but the quality of the data. It is common to see usage data where most of the transactions are just on a handful of popular items, and there is "little signal" for other items.

## Step 4 - Let's do some training!

Now that you have uploaded both the catalog and the usage data, we are ready to train the engine so that it can learn patterns from our data.

1. Click the **New Build** button.
2. Select **Recommendations** as the build type. Notice that we support a Ranking Build and an FBT (Frequently Bought Together) build types as well.



An FBT build allows you to identify patterns for products that are usually purchased/consumed in the same transaction. A ranking build is used to identify features of interest. We won't go very deep into FBT or ranking builds in this workshop, but if you are interested you should check out the [Build types and model quality documentation page](#).

3. Click the **Build** button. The build process takes about five minutes if you are using the Books data. It takes longer on larger datasets.

## Step 5 - Let's find out what the machine learned!

Once your build is completed, you will notice a new build in the builds list. This build can be queried for item and user recommendations.

1. Once your build is completed, click **Score**. This allows you to play with the model and see what items are recommended.

# MyFirstModel

Model ID: 89f8a410-87a3-4b9d-8c19-18acbc917210

The screenshot shows the Recommendations UI interface. It has three main sections: Step 1 (Add a catalog file), Step 2 (Add usage files), and Step 3 (Create a build). Step 1 and Step 2 each have a 'BROWSE FOR FILE...' button. Step 2 shows an uploaded file named 'usage\_large.txt'. Step 3 has a 'NEW BUILD' button and a table with one row. The table columns are BUILD ID, TYPE, STATUS, DATE CREATED, and ACTIVE BUILD. The row contains '1576770', 'Recommendation', 'Success', '10/3/2016 7:03:07 PM', and a radio button. A red box highlights the 'SCORE' button in the bottom right corner of the Step 3 section.

BUILD ID	TYPE	STATUS	DATE CREATED	ACTIVE BUILD
1576770	Recommendation	Success	10/3/2016 7:03:07 PM	<input type="radio"/>

1. Train a model by clicking the 'TRAIN MODEL' button.
2. Select an item to see which items are returned as recommendations for that item. Notice that if there are not enough transactions to predict a recommendation for a particular item, the system will not return any recommendations for that item. If for some reason you have an item that returns no recommendations, try scoring other items.

## Step 6 - Next steps

Congratulations! you have trained a model and then got recommendations from that model. The Recommendations UI is a useful tool that allows you to see the state of your projects and builds.

Now that you have a model, you may want to learn how to do all the steps above programmatically. In order to do learn to call the API programmatically, we invite you to check the [Recommendations API Reference](#) and download the [Recommendations Sample Application](#).

# Get recommendations in batches

1/17/2017 • 5 min to read • [Edit Online](#)

## NOTE

Getting recommendations in batches is more complicated than getting recommendations one at a time. Check the APIs for information about how to get recommendations for a single request:

[Item-to-Item recommendations](#)

[User-to-Item recommendations](#)

Batch scoring only works for builds that were created after July 21, 2016.

There are situations in which you need to get recommendations for more than one item at a time. For instance, you might be interested in creating a recommendations cache or even analyzing the types of recommendations that you are getting.

Batch scoring operations, as we call them, are asynchronous operations. You need to submit the request, wait for the operation to finish, and then gather your results.

To be more precise, these are the steps to follow:

1. Create an Azure Storage container if you don't have one already.
2. Upload an input file that describes each of your recommendation requests to Azure Blob storage.
3. Kick-start the scoring batch job.
4. Wait for the asynchronous operation to finish.
5. When the operation has finished, gather the results from Blob storage.

Let's walk through each of these steps.

## Create a Storage container if you don't have one already

Go to the [Azure portal](#) and create a new storage account if you don't have one already. To do this, navigate to **New > Data + Storage > Storage Account**.

After you have a storage account, you need to create the blob containers where you will store the input and output of the batch execution.

Upload an input file that describes each of your recommendation requests to Blob storage--let's call the file `input.json` here. After you have a container, you need to upload a file that describes each of the requests that you need to perform from the recommendations service.

A batch can perform only one type of request from a specific build. We will explain how to define this information in the next section. For now, let's assume that we will be performing item recommendations out of a specific build. The input file then contains the input information (in this case, the seed items) for each of the requests.

This is an example of what the `input.json` file looks like:

```
{
  "requests": [
    { "SeedItems": [ "C9F-00163", "FKF-00689" ] },
    { "SeedItems": [ "F34-03453" ] },
    { "SeedItems": [ "D16-3244" ] },
    { "SeedItems": [ "C9F-00163", "FKF-00689" ] },
    { "SeedItems": [ "F43-01467" ] },
    { "SeedItems": [ "BD5-06013" ] },
    { "SeedItems": [ "P45-00163", "FKF-00689" ] },
    { "SeedItems": [ "C9A-69320" ] }
  ]
}
```

As you can see, the file is a JSON file, where each of the requests has the information that's necessary to send a recommendations request. Create a similar JSON file for the requests that you need to fulfill, and copy it to the container that you just created in Blob storage.

## Kick-start the batch job

The next step is to submit a new batch job. For more information, check the [API reference](#).

The request body of the API needs to define the locations where the input, output, and error files need to be stored. It also needs to define the credentials that are necessary to access those locations. In addition, you need to specify some parameters that apply to the whole batch (the type of recommendations to request, the model/build to use, the number of results per call, and so on.)

This is an example of what the request body should look like:

```
{
  "input": {
    "authenticationType": "PublicOrSas",
    "baseLocation": "https://mystorage1.blob.core.windows.net/",
    "relativeLocation": "container1/batchInput.json",
    "sasBlobToken": "?sv=2015-07_restofToken_...4Z&sp=rw"
  },
  "output": {
    "authenticationType": "PublicOrSas",
    "baseLocation": "https://mystorage1.blob.core.windows.net/",
    "relativeLocation": "container1/batchOutput.json",
    "sasBlobToken": "?sv=2015-07_restofToken_...4Z&sp=rw"
  },
  "error": {
    "authenticationType": "PublicOrSas",
    "baseLocation": "https://mystorage1.blob.core.windows.net/",
    "relativeLocation": "container1/errors.txt",
    "sasBlobToken": "?sv=2015-07_restofToken_...4Z&sp=rw"
  },
  "job": {
    "apiName": "ItemRecommend",
    "modelId": "9ac12a0a-1add-4bdc-bf42-c6517942b3a6",
    "buildId": 1015703,
    "numberOfResults": 10,
    "includeMetadata": true,
    "minimalScore": 0.0
  }
}
```

Here a few important things to note:

- Currently, **authenticationType** should always be set to **PublicOrSas**.
- You need to get a Shared Access Signature (SAS) token to allow the Recommendations API to read and write

from/to your Blob storage account. More information about how to generate SAS tokens can be found on the [Recommendations API page](#).

- The only **apiName** that's currently supported is **ItemRecommend**, which is used for Item-to-Item recommendations. Batching doesn't currently support User-to-Item recommendations.

## Wait for the asynchronous operation to finish

When you start the batch operation, the response returns the Operation-Location header that gives you the information that's necessary to track the operation. You track the operation by using the [Retrieve Operation Status API](#), just like you do for tracking the operation of a build operation.

## Get the results

After the operation has finished, assuming that there were no errors, you can gather the results from your output Blob storage.

The example below show what the output might look like. In this example, we show results for a batch with only two requests (for brevity).

```
{
  "results": [
    {
      "request": { "seedItems": [ "DAF-00500", "P3T-00003" ] },
      "recommendations": [
        {
          "items": [
            {
              "itemId": "F5U-00011",
              "name": "L2 Ethernet Adapter-Win8Pro SC EN/XD/XX Hdwr",
              "metadata": ""
            }
          ],
          "rating": 0.722,
          "reasoning": [ "People who like the selected items also like 'L2 Ethernet Adapter-Win8Pro SC EN/XD/XX Hdwr'" ]
        },
        {
          "items": [
            {
              "itemId": "G5Y-00001",
              "name": "Docking Station for Srf Pro/Pro2 SC EN/XD/ES Hdwr",
              "metadata": ""
            }
          ],
          "rating": 0.718,
          "reasoning": [ "People who like the selected items also like 'Docking Station for Srf Pro/Pro2 SC EN/XD/ES Hdwr'" ]
        }
      ]
    },
    {
      "request": { "seedItems": [ "C9F-00163" ] },
      "recommendations": [
        {
          "items": [
            {
              "itemId": "C9F-00172",
              "name": "Nokia 2K Shell for Nokia Lumia 630/635 - Green",
              "metadata": ""
            }
          ],
          "rating": 0.649,
          "reasoning": [ "People who like the selected items also like 'Nokia 2K Shell for Nokia Lumia 630/635 - Green'" ]
        }
      ]
    }
  ]
}
```

```

    "reasoning": [ "People who like 'MOZO Flip Cover for Nokia Lumia 635 - White' also like 'Nokia 2K
Shell for Nokia Lumia 630/635 - Green'" ]
},
{
  "items": [
    {
      "itemId": "C9F-00171",
      "name": "Nokia 2K Shell for Nokia Lumia 630/635 - Orange",
      "metadata": ""
    }
  ],
  "rating": 0.647,
  "reasoning": [ "People who like 'MOZO Flip Cover for Nokia Lumia 635 - White' also like 'Nokia 2K
Shell for Nokia Lumia 630/635 - Orange'" ]
},
{
  "items": [
    {
      "itemId": "C9F-00170",
      "name": "Nokia 2K Shell for Nokia Lumia 630/635 - Yellow",
      "metadata": ""
    }
  ],
  "rating": 0.646,
  "reasoning": [ "People who like 'MOZO Flip Cover for Nokia Lumia 635 - White' also like 'Nokia 2K
Shell for Nokia Lumia 630/635 - Yellow'" ]
}
]
}
]}

```

## Learn about the limitations

- Only one batch job can be called per subscription at a time.
- A batch job input file cannot be more than 2 MB.

# Speaker Recognition API

4/12/2017 • 1 min to read • [Edit Online](#)

Welcome to the Microsoft Speaker Recognition APIs. Speaker Recognition APIs are cloud-based APIs that provide the most advanced algorithms for speaker verification and speaker identification. Speaker Recognition can be divided into two categories: speaker verification and speaker identification.

## Speaker Verification

Voice has unique characteristics that can be used to identify a person, just like a fingerprint. Using voice as a signal for access control and authentication scenarios has emerged as a new innovative tool –essentially offering a level up in security that simplifies the authentication experience for customers.

Speaker Verification APIs can automatically verify and authenticate users using their voice or speech.

### Enrollment

Enrollment for speaker verification is text-dependent, which means speakers need to choose a specific pass phrase to use during both enrollment and verification phases.

In enrollment, the speaker's voice is recorded saying a specific phrase, then a number of features are extracted and the chosen phrase is recognized. Together, both extracted features and the chosen phrase form a unique voice signature.

### Verification

#

In verification, an input voice and phrase are compared against the enrollment's voice signature and phrase –in order to verify whether or not they are from the same person, and if they are saying the correct phrase.

For more details about speaker verification, please refer to the API [Speaker - Verification](#).

## Speaker Identification

Speaker Identification APIs can automatically identify the person speaking in an audio file, given a group of prospective speakers. The input audio is paired against the provided group of speakers, and in the case that there is a match found, the speaker's identity is returned.

All speakers should go through an enrollment process first to get their voice registered to the system, and have a voice print created.

### Enrollment

Enrollment for speaker identification is text-independent, which means that there are no restrictions on what the speaker says in the audio. The speaker's voice is recorded, and a number of features are extracted to form a unique voice signature.

### Recognition

The audio of the unknown speaker, together with the prospective group of speakers, is provided during recognition. The input voice is compared against all speakers in order to determine whose voice it is, and if there is a match found, the identity of the speaker is returned.

For more details about speaker identification, please refer to the API [Speaker - Identification](#).

# Text Analytics Documentation

4/12/2017 • 1 min to read • [Edit Online](#)

## Description

Understanding and analyzing unstructured text is an increasingly popular field and includes a wide spectrum of problems such as sentiment analysis, key phrase extraction, topic modeling/extraction, aspect extraction and more.

Text Analytics API is a suite of text analytics services built with Azure Machine Learning. We currently offer APIs for sentiment analysis, key phrase extraction and topic detection for English text, as well as language detection for 120 languages. In this initial preview release, we offer APIs for sentiment analysis and key phrase extraction of English text. No labeled or training data is needed to use the service - just bring your text data. This service is based on research and engineering that originated in Microsoft Research and which has been battle-tested and improved over the past few years by product teams such as Bing and Office.

## Sentiment Analysis

Let's say you run a website to sell handicrafts. Your users submit feedback on your site, and you'd like to find out what users think of your brand, and how that changes over time as you release new products and features to your site. Sentiment analysis can help here - given a piece of text, the Azure ML Text Analytics service returns a score between 0 and 1 denoting overall sentiment in the input text. Scores close to 1 indicate positive sentiment, while scores close to 0 indicate negative sentiment.

Sentiment score is generated using classification techniques. The input features to the classifier include n-grams, features generated from part-of-speech tags, and embedded words. The classifier was trained in part using Sentiment140 data.

## Key Phrase Extraction

This service can also extract key phrases, which denote the main talking points in the text. We employ techniques from Microsoft Office's sophisticated Natural Language Processing toolkit.

For example, for the input text 'The manual transmission is a bit twitchy. Also, the vehicle is old-school', the service would return the main talking points: 'manual transmission', 'vehicle' and 'old-school'.

## Topic Detection

This is a new service which returns the topics which have been detected in multiple text articles. The service is designed to work well for short, human written text such as reviews and user feedback, and can help you to understand the main issues or suggestions that customers are mentioning.

## Language Detection

The service can be used to detect which language the input text is written in.

# Getting started with the Text Analytics APIs to detect sentiment, key phrases, topics and language

1/17/2017 • 7 min to read • [Edit Online](#)

This document describes how to onboard your service or application to use the [Text Analytics APIs](#). You can use these APIs to detect sentiment, key phrases, topics and language from your text. [Click here to see an interactive demo of the experience.](#)

Please refer to the [API definitions](#) for technical documentation for the APIs.

This guide is for version 2 of the APIs. For details on version 1 of the APIs, [refer to this document](#).

By the end of this tutorial, you will be able to programatically detect:

- **Sentiment** - Is text positive or negative?
- **Key phrases** - What are people discussing in a single article?
- **Topics** - What are people discussing across many articles?
- **Languages** - What language is text written in?

Note that this API charges 1 transaction per document submitted. As an example, if you request sentiment for 1000 documents in a single call, 1000 transactions will be deducted.

## General overview

This document is a step-by-step guide. Our objective is to walk you through the steps necessary to train a model, and to point you to resources that will allow you to put it in production. This exercise will take about 30 minutes.

For these tasks, you will need an editor and call the RESTful endpoints in your language of choice.

Let's get started!

## Task 1 - Signing up for the Text Analytics APIs

In this task, you will sign up for the text analytics service.

1. Navigate to **Cognitive Services** in the [Azure Portal](#) and ensure **Text Analytics** is selected as the 'API type'.
2. Select a plan. You may select the **free tier for 5,000 transactions/month**. As is a free plan, you will not be charged for using the service. You will need to login to your Azure subscription.
3. Complete the other fields and create your account.
4. After you sign up for Text Analytics, find your **API Key**. Copy the primary key, as you will need it when using the API services.

## Task 2 - Detect sentiment, key phrases and languages

It's easy to detect sentiment, key phrases and languages in your text. You will programatically get the same results as the [demo experience](#) returns.

**TIP**

For sentiment analysis, we recommend that you split text into sentences. This generally leads to a higher precision in sentiment predictions.

Note that the supported languages are as follows:

FEATURE	SUPPORTED LANGUAGE CODES
Sentiment	<code>en</code> (English), <code>es</code> (Spanish), <code>fr</code> (French), <code>pt</code> (Portuguese)
Key phrases	<code>en</code> (English), <code>es</code> (Spanish), <code>de</code> (German), <code>ja</code> (Japanese)

1. You will need to set the headers to the following. Note that JSON is currently the only accepted input format for the APIs. XML is not supported.

```
Ocp-Apim-Subscription-Key: <your API key>
Content-Type: application/json
Accept: application/json
```

2. Next, format your input rows in JSON. For sentiment, key phrases and language, the format is the same. Note that each ID should be unique and will be the ID returned by the system. The maximum size of a single document that can be submitted is 10KB, and the total maximum size of submitted input is 1MB. No more than 1,000 documents may be submitted in one call. Rate limiting exists at a rate of 100 calls per minute - we therefore recommend that you submit large quantities of documents in a single call. Language is an optional parameter that should be specified if analyzing non-English text. An example of input is shown below, where the optional parameter `language` for sentiment analysis or key phrase extraction is included:

```
{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "First document"
    },
    ...
    {
      "language": "en",
      "id": "100",
      "text": "Final document"
    }
  ]
}
```

3. Make a **POST** call to the system with the input for sentiment, key phrases and language. The URLs will look as follows:

```
POST https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment
POST https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/keyPhrases
POST https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/languages
```

4. This call will return a JSON formatted response with the IDs and detected properties. An example of the output for sentiment is shown below (with error details excluded). In the case of sentiment, a score between 0 and 1 will be returned for each document:

```

// Sentiment response
{
  "documents": [
    {
      "id": "1",
      "score": "0.934"
    },
    ...
    {
      "id": "100",
      "score": "0.002"
    },
  ],
}

// Key phrases response
{
  "documents": [
    {
      "id": "1",
      "keyPhrases": ["key phrase 1", ..., "key phrase n"]
    },
    ...
    {
      "id": "100",
      "keyPhrases": ["key phrase 1", ..., "key phrase n"]
    },
  ]
}

// Languages response
{
  "documents": [
    {
      "id": "1",
      "detectedLanguages": [
        {
          "name": "English",
          "iso6391Name": "en",
          "score": "1"
        }
      ]
    },
    ...
    {
      "id": "100",
      "detectedLanguages": [
        {
          "name": "French",
          "iso6391Name": "fr",
          "score": "0.985"
        }
      ]
    }
  ]
}

```

## Task 3 - Detect topics in a corpus of text

This is a newly released API which returns the top detected topics for a list of submitted text records. A topic is identified with a key phrase, which can be one or more related words. The API is designed to work well for short, human written text such as reviews and user feedback.

This API requires a **minimum of 100 text records** to be submitted, but is designed to detect topics across

hundreds to thousands of records. Any non-English records or records with less than 3 words will be discarded and therefore will not be assigned to topics. For topic detection, the maximum size of a single document that can be submitted is 30KB, and the total maximum size of submitted input is 30MB. Topic detection is rate limited to 5 submissions every 5 minutes.

There are two additional **optional** input parameters that can help to improve the quality of results:

- **Stop words.** These words and their close forms (e.g. plurals) will be excluded from the entire topic detection pipeline. Use this for common words (for example, "issue", "error" and "user" may be appropriate choices for customer complaints about software). Each string should be a single word.
- **Stop phrases** - These phrases will be excluded from the list of returned topics. Use this to exclude generic topics that you don't want to see in the results. For example, "Microsoft" and "Azure" would be appropriate choices for topics to exclude. Strings can contain multiple words.

Follow these steps to detect topics in your text.

1. Format the input in JSON. This time, you can define stop words and stop phrases.

```
{  
    "documents": [  
        {  
            "id": "1",  
            "text": "First document"  
        },  
        ...  
        {  
            "id": "100",  
            "text": "Final document"  
        }  
    ],  
    "stopWords": [  
        "issue", "error", "user"  
    ],  
    "stopPhrases": [  
        "Microsoft", "Azure"  
    ]  
}
```

2. Using the same headers as defined in Task 2, make a **POST** call to the topics endpoint:

```
POST https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/topics
```

3. This will return an `operation-location` as the header in the response, where the value is the URL to query for the resulting topics:

```
'operation-location':  
'https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/operations/<operationId>'
```

4. Query the returned `operation-location` periodically with a **GET** request. Once per minute is recommended.

```
GET https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/operations/<operationId>
```

5. The endpoint will return a response including `{"status": "notstarted"}` before processing, `{"status": "running"}` while processing and `{"status": "succeeded"}` with the output once completed. You can then consume the output which will be in the following format (note details like error format and dates have been excluded from this example):

```
{
    "status": "succeeded",
    "operationProcessingResult": {
        "topics": [
            {
                "id": "8b89dd7e-de2b-4a48-94c0-8e7844265196",
                "score": "5",
                "keyPhrase": "first topic name"
            },
            ...
            {
                "id": "359ed9cb-f793-4168-9cde-cd63d24e0d6d",
                "score": "3",
                "keyPhrase": "final topic name"
            }
        ],
        "topicAssignments": [
            {
                "topicId": "8b89dd7e-de2b-4a48-94c0-8e7844265196",
                "documentId": "1",
                "distance": "0.354"
            },
            ...
            {
                "topicId": "359ed9cb-f793-4168-9cde-cd63d24e0d6d",
                "documentId": "55",
                "distance": "0.758"
            },
        ]
    }
}
```

Note that the successful response for topics from the `operations` endpoint will have the following schema:

```
{
    "topics" : [
        {
            "id" : "string",
            "score" : "number",
            "keyPhrase" : "string"
        }],
    "topicAssignments" : [
        {
            "documentId" : "string",
            "topicId" : "string",
            "distance" : "number"
        }],
    "errors" : [
        {
            "id" : "string",
            "message" : "string"
        }]
}
```

Explanations for each part of this response are as follows:

### topics

KEY	DESCRIPTION
id	A unique identifier for each topic.
score	Count of documents assigned to topic.

KEY	DESCRIPTION
keyPhrase	A summarizing word or phrase for the topic.

## topicAssignments

KEY	DESCRIPTION
documentId	Identifier for the document. Equates to the ID included in the input.
topicId	The topic ID which the document has been assigned to.
distance	Document-to-topic affiliation score between 0 and 1. The lower a distance score the stronger the topic affiliation is.

## errors

KEY	DESCRIPTION
id	Input document unique identifier the error refers to.
message	Error message.

## Next steps

Congratulations! You have now completed using text analytics on your data. You may now wish to look into using a tool such as [Power BI](#) to visualize your data, as well as automating your insights to give you a real-time view of your text data.

To see how Text Analytics capabilities, such as sentiment, can be used as part of a bot, see the [Emotional Bot](#) example on the Bot Framework site.

# Upgrading to Version 2 of the Text Analytics API

1/17/2017 • 3 min to read • [Edit Online](#)

This guide will take you through the process of upgrading your code from using the [first version of the API](#) to using the second version.

If you have not used the API and would like to learn more, you can [learn more about the API here](#) or [follow the Quick Start Guide](#). For technical reference, refer to the [API Definition](#).

## Part 1. Get a new key

First, you will need to get a new API key from the [Azure Portal](#):

1. Navigate to the Text Analytics service through the [Cortana Intelligence Gallery](#). Here, you will also find links to the documentation and code samples.
2. Click **Sign Up**. This link will take you to the Azure management portal, where you can sign up for the service.
3. Select a plan. You may select the **free tier for 5,000 transactions/month**. As is a free plan, you will not be charged for using the service. You will need to login to your Azure subscription.
4. After you sign up for Text Analytics, you'll be given an **API Key**. Copy this key, as you'll need it when using the API services.

## Part 2. Update the headers

Update the submitted header values as shown below. Note that the account key is no longer encoded.

### Version 1

```
Authorization: Basic base64encode(<your Data Market account key>)
Accept: application/json
```

### Version 2

```
Content-Type: application/json
Accept: application/json
Ocp-Apim-Subscription-Key: <your Azure Portal account key>
```

## Part 3. Update the base URL

### Version 1

```
https://api.datamarket.azure.com/data.ashx/aml/text-analytics/v1/
```

### Version 2

```
https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/
```

## Part 4a. Update the formats for sentiment, key phrases and languages

### Endpoints

GET endpoints have now been deprecated, so all input should be submitted as a POST request. Update the endpoints to the ones shown below.

	<b>VERSION 1 SINGLE ENDPOINT</b>	<b>VERSION 1 BATCH ENDPOINT</b>	<b>VERSION 2 ENDPOINT</b>
Call type	GET	POST	POST
Sentiment	<code>GetSentiment</code>	<code>GetSentimentBatch</code>	<code>sentiment</code>
Key phrases	<code>GetKeyPhrases</code>	<code>GetKeyPhrasesBatch</code>	<code>keyPhrases</code>
Languages	<code>GetLanguage</code>	<code>GetLanguageBatch</code>	<code>languages</code>

#### Input formats

Note that only POST format is now accepted, so you should reformat any input which previously used the single document endpoints accordingly. Inputs are not case sensitive.

#### Version 1 (batch)

```
{
  "Inputs": [
    {
      "Id": "string",
      "Text": "string"
    }
  ]
}
```

#### Version 2

```
{
  "documents": [
    {
      "id": "string",
      "text": "string"
    }
  ]
}
```

#### Output from sentiment

##### Version 1

```
{
  "SentimentBatch": [{{
    "Id": "string",
    "Score": "double"
  }}],
  "Errors": [{{
    "Id": "string",
    "Message": "string"
  }}]
}
```

##### Version 2

```
{  
  "documents": [{  
    "id": "string",  
    "score": "double"  
  }],  
  "errors" : [{  
    "id": "string",  
    "message": "string"  
  }]  
}
```

#### Output from key phrases

##### Version 1

```
{  
  "KeyPhrasesBatch": [{  
    "Id": "string",  
    "KeyPhrases": ["string"]  
  }],  
  "Errors" : [{  
    "Id": "string",  
    "Message": "string"  
  }]  
}
```

##### Version 2

```
{  
  "documents": [{  
    "id": "string",  
    "keyPhrases": ["string"]  
  }],  
  "errors" : [{  
    "id": "string",  
    "message": "string"  
  }]  
}
```

#### Output from languages

##### Version 1

```
{  
  "LanguageBatch": [{  
    "id": "string",  
    "detectedLanguages": [{  
      "Score": "double",  
      "Name": "string",  
      "Iso6391Name": "string"  
    }]  
  }],  
  "Errors" : [{  
    "Id": "string",  
    "Message": "string"  
  }]  
}
```

##### Version 2

```
{
  "documents": [
    {
      "id": "string",
      "detectedLanguages": [
        {
          "score": "double",
          "name": "string",
          "iso6391Name": "string"
        }
      ]
    },
    "errors": [
      {
        "id": "string",
        "message": "string"
      }
    ]
  }
}
```

## Part 4b. Update the formats for topics

### Endpoints

	VERSION 1 ENDPOINT	VERSION 2 ENDPOINT
Submit for topic detection (POST)	StartTopicDetection	topics
Fetch topic results (GET)	GetTopicDetectionResult?JobId=<jobId>	operations/<operationId>

### Input formats

#### Version 1

```
{
  "StopWords": [
    "string"
  ],
  "StopPhrases": [
    "string"
  ],
  "Inputs": [
    {
      "Id": "string",
      "Text": "string"
    }
  ]
}
```

#### Version 2

```
{
  "stopWords": [
    "string"
  ],
  "stopPhrases": [
    "string"
  ],
  "documents": [
    {
      "id": "string",
      "text": "string"
    }
  ]
}
```

### Submission results

## Version 1 (POST)

Previously, when the job finished, you would receive the following JSON output, where the jobId would be appended to a URL to fetch the output.

```
{  
    "odata.metadata": "<url>",  
    "JobId": "<JobId>"  
}
```

## Version 2 (POST)

The response will now include a header value as follows, where `operation-location` is used as the endpoint to poll for the results:

```
'operation-location': 'https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/operations/<operationId>'
```

## Operation results

### Version 1 (GET)

```
{  
    "TopicInfo" : [ {  
        "TopicId" : "string"  
        "Score" : "double"  
        "KeyPhrase" : "string"  
    } ],  
    "TopicAssignment" : [ {  
        "Id" : "string",  
        "TopicId" : "string",  
        "Distance" : "double"  
    } ],  
    "Errors" : [ {  
        "Id": "string",  
        "Message": "string"  
    } ]  
}
```

### Version 2 (GET)

As before, **periodically poll the output** (the suggested period is every minute) until the output is returned.

When the topics API has finished, a status reading `succeeded` will be returned. This will then include the output results in the format shown below:

```
{  
    "status": "succeeded",  
    "createdDateTime": "string",  
    "operationType": "topics",  
    "processingResult": {  
        "topics" : [{  
            "id" : "string"  
            "score" : "double"  
            "keyPhrase" : "string"  
        }],  
        "topicAssignments" : [{  
            "topicId" : "string",  
            "documentId" : "string",  
            "distance" : "double"  
        }],  
        "errors" : [{  
            "id": "string",  
            "message": "string"  
        }]  
    }  
}
```

## Part 5. Test it!

You should now be good to go! Test your code with a small sample to ensure that you can successfully process your data.

# Microsoft Translator API

4/12/2017 • 1 min to read • [Edit Online](#)

Microsoft Translator APIs can be seamlessly integrated into your applications, websites, tools, or other solutions to provide multi-language user experiences. Leveraging industry standards, it can be used on any hardware platform and with any operating system to perform language translation and other language-related operations such as speech to speech translation, text language detection or text to speech.

## Microsoft Translator Deep Neural Network translation

Microsoft Translator API has been using Statistical Machine Translation (SMT) technology since it started. The technology has reached a plateau in terms of performance improvement. Translation quality is no longer improving with SMT. A new AI-based translation technology is gaining momentum based on Deep Neural Networks (DNN).

DNN enabled translations will first appear for users of the Microsoft Translator Speech API, and will spread from there to eventually include all languages and APIs.

## Technology

DNN models are at the core of the API and are not visible to end users. The only noticeable differences will be:

- The improved translation quality, especially for languages such as Chinese and Japanese.
- The incompatibility with the existing Hub and CTF customization features

## High-level product architecture

### How DNN works

DNN will provide better translations not only from a raw translation quality scoring standpoint but also because they will sound more fluid, more human, than SMT ones. The key reason for this fluidity is that DNN translation uses the full context of a sentence to translate words contrary to SMT that only takes the context of a few words before and after each word. Use cases Although DNN translation can be used for all types of translations the improvements are particularly impressive for languages where translation quality has previously lagged. For instance, Japanese, Chinese and Arabic.

The Microsoft Translator Hub service and API lets you customize translations for words and sentences that are specific to a particular domain of knowledge.

The Microsoft Translator API also offers the unique ability to improve the accuracy of the delivered translations through the use of the Collaborative Translations Framework (CTF), allowing users to recommend alternative translations to those provided by Translator's automatic translation engine.

To get started, you will first need to [sign up for a subscription](#) to the text or speech API. Free tiers are available for 2 million characters per month for the text API and 2 hours per month for the speech API.

### Solution and pricing information.

# Supported languages

4/17/2017 • 1 min to read • [Edit Online](#)

There are three groups of supported languages in the Microsoft Translator API. Text, Speech, and Text-to-speech (TTS). The languages API method returns the list of supported languages for each of the three groups. The languages method does not require an access token for authentication.

In the 'Scope' parameter of the method, add one or all three of the groups. After you have entered the scope parameters, select the **Try it out!** button. The service returns the supported languages in JSON format. The response is visible in the 'Response Body' section.

[Visit the API reference to try out the languages method.](#)

[See the list of languages on our web site.](#)

# Your world in your words

4/12/2017 • 1 min to read • [Edit Online](#)

With Microsoft Translator Hub you can build your own machine translation system using your preferred terminology and style specific to your industry, linguistic, domain, or organizational needs. You can use this system via the Microsoft Translator API in your own applications, in 3rd party applications, your website, the Bing Translator Widget, and a community of collaborators.

[Click here for more information about the Hub](#)

# Collaborative Translation Framework (CTF) Reporting API

4/12/2017 • 14 min to read • [Edit Online](#)

The Collaborative Translation Framework (CTF) Reporting API returns statistics and the actual content in the CTF store. This API is different from the GetTranslations() method because it:

- Returns the translated content and its total count only from your account (appId or Azure Marketplace account).
- Returns the translated content and its total count without requiring a match of the source sentence.
- Does not return the automatic translation (machine translation).

**Endpoint** The endpoint of the CTF Reporting API is

<http://api.microsofttranslator.com/v2/beta/ctfreporting.svc>

## Methods

NAME	DESCRIPTION
GetUserTranslationCounts Method	Get counts of the translations that are created by the user
GetUserTranslations Method	Retrieves the translations that are created by the user.

**Remarks** These methods enable you to:

- Retrieve the complete set of user translations and corrections under your account ID for download.
- Obtain the list of the frequent contributors. Ensure that the correct user name is provided in AddTranslation().
- Build a user interface (UI) that allows your trusted users to see all available candidates, if necessary restricted to a portion of your site, based on the URI prefix.

### NOTE

Both the methods are relatively slow and expensive. It is recommended to use them sparingly.

## GetUserTranslations Method

This method retrieves the translations that are created by the user. It provides the translations grouped by the uriPrefix, from, to, user, and minRating and maxRating request parameters.

Syntax

### C#

```
UserTranslation[] GetUserTranslations ( string appId, string uriPrefix, string from, string to, int? minRating, int? maxRating, string user, string category DateTime? minDateUtc, DateTime? maxDateUtc, int? skip, int? take);
```

## Parameters

PARAMETER	DESCRIPTION
-----------	-------------

PARAMETER	DESCRIPTION
appId	<b>Required.</b> If the Authorization header is used, leave the appId field empty else specify a string containing "Bearer" + " " + access token.
uriPrefix	<b>Optional.</b> A string containing prefix of URI of the translation.
from	<b>Optional.</b> A string representing the language code of the translation text.
to	<b>Optional.</b> A string representing the language code to translate the text into.
minRating	<b>Optional.</b> An integer value representing the minimum quality rating for the translated text. The valid value is between -10 and 10. The default value is 1.
maxRating	<b>Optional.</b> An integer value representing the maximum quality rating for the translated text. The valid value is between -10 and 10. The default value is 1.
user	<b>Optional.</b> A string that is used to filter the result based on the originator,of the submission.
category	<b>Optional.</b> A string containing the category or domain of the translation,.This parameter supports only the default option general.
minDateUtc	<b>Optional.</b> The date from when you want to retrieve the translations. The date must be in the UTC format.
maxDateUtc	<b>Optional.</b> The date till when you want to retrieve the translations. The date must be in the UTC format.
skip	<b>Optional.</b> The number of results that you want to skip on a page. For example, if you want to skip the first 20 rows of the results and view from the 21st result record, specify 20 for this parameter. The default value for this parameter is 0.
take	<b>Optional.</b> The number of results that you want to retrieve. The maximum number of each request is 100. The default is 50.

#### NOTE

The skip and take request parameters enable pagination for a large number of result records.

#### Return Value

The result set contains array of the **UserTranslation**. Each UserTranslation has the following elements:

FIELD	DESCRIPTION
CreatedDateUtc	The creation date of the entry using AddTranslation().

FIELD	DESCRIPTION
From	The source language
OriginalText	The source language text that is used when submitting the request.
Rating	The rating that is applied by the submitter in the AddTranslation() method call.
To	The target language
TranslatedText	The translation as submitted in the AddTranslation() method call.
Uri	The URI applied in the AddTranslation() method call.
User	The user name

## Exceptions

EXCEPTION	MESSAGE	CONDITIONS
ArgumentOutOfRangeException	The parameter ' <b>maxDateUtc</b> ' must be greater than or equal to ' <b>minDateUtc</b> '.	The value of the parameter <b>maxDateUtc</b> is lesser than the value of the parameter <b>minDateUtc</b> .
TranslateApiException	IP is over the quota.	The limit for the number of requests per minute is reached. The request size remains limited at 10000 characters. An hourly and a daily quota limit the number of characters that the Microsoft Translator API will accept.
	AppId is over the quota.	The application ID exceeded the hourly or daily quota.

### NOTE

The quota will adjust to ensure fairness among all users of the service.

## Example C# PHP

### C#

```
using System;
using System.IO;
using System.Net;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Web;

namespace CtfReporting
{
    class Program
    {
```

```

static void Main(string[] args)
{
    AdmAccessToken admToken;
    string headerValue;
    //Get Client Id and Client Secret from https://datamarket.azure.com/developer/applications/
    //Refer obtaining AccessToken (http://msdn.microsoft.com/en-us/library/hh454950.aspx)
    AdmAuthentication admAuth = new AdmAuthentication("clientId", "clientSecret");
    try
    {
        admToken = admAuth.GetAccessToken();
        // Create a header with the access_token property of the returned token
        headerValue = "Bearer " + admToken.access_token;
        Console.WriteLine("User Translations");
        ctf GetUserTranslations(headerValue, 0, 5);
        Console.WriteLine("Press Enter for Next 5 Records");
        if (Console.ReadKey().Key == ConsoleKey.Enter)
        {
            ctf GetUserTranslations(headerValue, 5, 5);
            Console.WriteLine("Press any key to exit...");
            Console.ReadKey(true);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private static void ctf GetUserTranslations(string authToken, int skip, int take)
{
    // Add CtfReportingService as a service reference,
Address:http://api.microsofttranslator.com/v2/beta/ctfreporting.svc
    CtfReportingService.CtfReportingServiceClient reportingClient = new
CtfReportingService.CtfReportingServiceClient();
    CtfReportingService.UserTranslation[] userTranslations =
reportingClient.GetUserTranslations(authToken, "", "es", "en", null, null, "", "general", null, null, skip,
take);
    foreach (CtfReportingService.UserTranslation item in userTranslations)
    {
        Console.WriteLine(string.Format("CreatedDateUtc={0},From={1},To={2},Rating={3},Uri={4},User=
{5},OriginalText={6},TranslatedText={7}", item.CreatedDateUtc, item.From, item.To, item.Rating, item.Uri,
item.User, item.OriginalText, item.TranslatedText));
    }
}

[DataContract]
public class AdmAccessToken
{
    [DataMember]
    public string access_token { get; set; }
    [DataMember]
    public string token_type { get; set; }
    [DataMember]
    public string expires_in { get; set; }
    [DataMember]
    public string scope { get; set; }
}

public class AdmAuthentication
{
    public static readonly string DatamarketAccessUri =
"https://datamarket.accesscontrol.windows.net/v2/OAuth2-13";
    private string clientId;
    private string clientSecret;
    private string request;

    public AdmAuthentication(string clientId, string clientSecret)
}

```

```

    {
        this.clientId = clientId;
        this.clientSecret = clientSecret;
        //If clientid or client secret has special characters, encode before sending request
        this.request = string.Format("grant_type=client_credentials&client_id={0}&client_secret={1}&scope=http://api.microsofttranslator.com", HttpUtility.UrlEncode(clientId),
        HttpUtility.UrlEncode(clientSecret));
    }

    public AdmAccessToken GetAccessToken()
    {
        return HttpPost(DatamarketAccessUri, this.request);
    }

    private AdmAccessToken HttpPost(string DatamarketAccessUri, string requestDetails)
    {
        //Prepare OAuth request
        WebRequest webRequest = WebRequest.Create(DatamarketAccessUri);
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.Method = "POST";
        byte[] bytes = Encoding.ASCII.GetBytes(requestDetails);
        webRequest.ContentLength = bytes.Length;
        using (Stream outputStream = webRequest.GetRequestStream())
        {
            outputStream.Write(bytes, 0, bytes.Length);
        }
        using (WebResponse webResponse = webRequest.GetResponse())
        {
            DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(AdmAccessToken));
            //Get deserialized object from JSON stream
            AdmAccessToken token = (AdmAccessToken)serializer.ReadObject(webResponse.GetResponseStream());
            return token;
        }
    }
}
```

```

\*\*PHP\*\*

```

<?php

```

```

class AccessTokenAuthentication {
    /*
     * Get the access token.
     *
     * @param string $grantType      Grant type.
     * @param string $scopeUrl       Application Scope URL.
     * @param string $clientID       Application client ID.
     * @param string $clientSecret   Application client ID.
     * @param string $authUrl        Oauth Url.
     *
     * @return string.
     */
    function getTokens($grantType, $scopeUrl, $clientID, $clientSecret, $authUrl){
        try {
            //Initialize the Curl Session.
            $ch = curl_init();
            //Create the request Array.
            $paramArr = array (
                'grant_type'      => $grantType,
                'scope'           => $scopeUrl,
                'client_id'       => $clientID,
                'client_secret'   => $clientSecret
            );
            //Create an Http Query.//
            $paramArr = http_build_query($paramArr);

```

```

        //Set the Curl URL.
        curl_setopt($ch, CURLOPT_URL, $authUrl);
        //Set HTTP POST Request.
        curl_setopt($ch, CURLOPT_POST, TRUE);
        //Set data to POST in HTTP "POST" Operation.
        curl_setopt($ch, CURLOPT_POSTFIELDS, $paramArr);
        //CURLOPT_RETURNTRANSFER- TRUE to return the transfer as a string of the return value of
curl_exec().
        curl_setopt ($ch, CURLOPT_RETURNTRANSFER, TRUE);
        //CURLOPT_SSL_VERIFYPeer- Set FALSE to stop cURL from verifying the peer's certificate.
        curl_setopt($ch, CURLOPT_SSL_VERIFYPeer, false);
        //Execute the cURL session.
        $strResponse = curl_exec($ch);
        //Get the Error Code returned by Curl.
        $curl_errno = curl_errno($ch);
        if($curl_errno){
            $curlError = curl_error($ch);
            throw new Exception($curlError);
        }
        //Close the Curl Session.
        curl_close($ch);
        //Decode the returned JSON string.
        $objResponse = json_decode($strResponse);

        if ($objResponse->error){
            throw new Exception($objResponse->error_description);
        }
        return $objResponse->access_token;
    } catch (Exception $e) {
        echo "Exception-".$e->getMessage();
    }
}

/*
 * Class:AccessTokenAuthentication
 *
 * Create SOAP Object.
 */
class SOAPMicrosoftTranslator {
    /*
     * Soap Object.
     *
     * @var ObjectArray.
     */
    public $objSoap;
    /*
     * Create the SAOP object.
     *
     * @param string $accessToken Access Token string.
     * @param string $wsdlUrl      WSDL string.
     *
     * @return string.
     */
    public function __construct($accessToken, $wsdlUrl){
        try {
            //Authorization header string.
            $authHeader = "Authorization: Bearer ". $accessToken;
            $contextArr = array(
                'http'  => array(
                    'header' => $authHeader
                )
            );
            //Create a streams context.
            $objContext = stream_context_create($contextArr);
            $optionsArr = array (
                'soap_version'  => 'SOAP_1_2',
                'encoding'       => 'UTF-8',
                'exceptions'    => true,

```

```

        'trace'          => true,
        'cache_wsdl'    => 'WSDL_CACHE_NONE',
        'stream_context' => $objContext,
        'user_agent'     => 'PHP-SOAP/'.PHP_VERSION."\r\n".$authHeader
    );
    //Call Soap Client.
    $this->objSoap = new SoapClient($wsdlUrl, $optionsArr);
} catch(Exception $e){
    echo "<h2>Exception Error!</h2>";
    echo $e->getMessage();
}
}

try {
//Soap WSDL Url.
$wsdlUrl      = "http://api.microsofttranslator.com/v2/beta/ctfreporting.svc";
//Client ID of the application.
$clientID      = "clientid";
//Client Secret key of the application.
$clientSecret = "clientsecret";
//OAuth Url.
$authUrl       = "https://datamarket.accesscontrol.windows.net/v2/OAuth2-13/";
//Application Scope Url
$scopeUrl      = "http://api.microsofttranslator.com";
//Application grant type
$grantType     = "client_credentials";

//Create the Authentication object
$authObj       = new AccessTokenAuthentication();
//Get the Access token
$accessToken   = $authObj->getTokens($grantType, $scopeUrl, $clientID, $clientSecret, $authUrl);
//Create soap translator Object
$soapTranslator = new SOAPMicrosoftTranslator($accessToken, $wsdlUrl);

//Set the Params.
$from          = '';
$to            = '';
$minRating    = 0;
$maxRating    = 10;
$takeResult   = 10;
$user          = '';

//Request argument list.
$requestArg = array (
    'user'         => $user,
    'from'         => $from,
    'to'           => $to,
    'minRating'   => $minRating,
    'maxRating'   => $maxRating,
    'take'         => $takeResult
);
//Call the GetUserTranslations Method.
$responseObj = $soapTranslator->objSoap-> GetUserTranslations($requestArg);
$translationArr = $responseObj-> GetUserTranslationsResult->UserTranslation;
if(sizeof($translationArr) > 0) {
    echo "<table border=2px>";
    echo "<tr>";
    echo "<td><b>User</b></td><td><b>From</b></td><td><b>To</b></td>
        <td><b>Rating</b></td><td><b>OriginalText</b></td><td><b>OriginalText</b></td>" ;
    echo "</tr>";
    if(sizeof($translationArr) > 1) {
        $i=1;
        foreach ($translationArr as $translation) {
            echo "<tr><td>$translation->User</td><td>$translation->From</td>
                <td>$translation->To</td><td>$translation->Rating</td>
                <td>$translation->OriginalText</td><td>$translation->TranslatedText</td></tr>";
        $i++;
    }
}

```

```

        }
    } else {
        echo "<tr><td>$translationArr->User</td><td>$translationArr->From</td>
        <td>$translationArr->To</td><td>$translationArr->Rating</td>
        <td>$translationArr->OriginalText</td><td>$translationArr->TranslatedText</td></tr>";
    }
}
echo "</table>";
} catch (Exception $e) {
    echo "Exception: " . $e->getMessage() . PHP_EOL;
}

```

## GetUserTranslationCounts Method

This method gets the count of translations that are created by the user. It provides the list of translation counts grouped by the uriPrefix, from, to, user, minRating, and maxRating request parameters.

### Syntax

#### C###

```
UserTranslationCount[] GetUserTranslationCounts( string appId, string uriPrefix, string from, string to, int? minRating, int? maxRating, string user, string category DateTime? minDateUtc, DateTime? maxDateUtc, int? skip, int? take);
```

### Parameters

| PARAMETER | DESCRIPTION                                                                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| appId     | Required. If the Authorization header is used, leave the appId field empty, else specify a string containing "Bearer" + " " + access token.                |
| uriPrefix | Optional. A string containing prefix of URI of the translation.                                                                                            |
| from      | Optional. A string representing the language code of the translation text.                                                                                 |
| to        | Optional. A string representing the language code to translate the text into.                                                                              |
| minRating | Optional. An integer value representing the minimum quality rating for the translated text. The valid value is between -10 and 10. The default value is 1. |
| maxRating | Optional. An integer value representing the maximum quality rating for the translated text. The valid value is between -10 and 10. The default value is 1. |
| user      | Optional. A string that is used to filter the result based on the originator of the submission.                                                            |
| category  | Optional. A string containing the category or domain of the translation. This parameter supports only the default option general.                          |

| PARAMETER  | DESCRIPTION                                                                                                                                                                                                                                        |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| minDateUtc | Optional. The date from when you want to retrieve the translations. The date must be in the UTC format.                                                                                                                                            |
| maxDateUtc | Optional. The date till when you want to retrieve the translations. The date must be in the UTC format.                                                                                                                                            |
| skip       | Optional. The number of results that you want to skip on a page. For example, if you want to skip the first 20 rows of the results and view from the 21st result record, specify 20 for this parameter. The default value for this parameter is 0. |
| take       | Optional. The number of results that you want to retrieve. The maximum number of each request is 100. The default is 100.                                                                                                                          |

Note: The skip and take request parameters enable pagination for a large number of result records.

## Return value

The result set contains array of the **UserTranslationCount**. Each UserTranslationCount has the following elements:

| FIELD  | DESCRIPTION                                                                      |
|--------|----------------------------------------------------------------------------------|
| Count  | The number of results that is retrieved.                                         |
| From   | The source language                                                              |
| Rating | The rating that is applied by the submitter in the AddTranslation() method call. |
| To     | The target language.                                                             |
| Uri    | The URI applied in the AddTranslation() method call.                             |
| USer   | The user name                                                                    |

## Exceptions

| EXCEPTION                    | MESSAGE                                                                                     | CONDITIONS                                                                                                                                                                                                                  |
|------------------------------|---------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ArgumentsOutOfRangeException | The parameter ' <b>maxDateUtc</b> ' must be greater than or equal to ' <b>minDateUtc</b> '. | The value of the parameter <b>maxDateUtc</b> is lesser than the value of the parameter <b>minDateUtc</b> .                                                                                                                  |
| TranslateApiException        | IP is over the quota.                                                                       | The limit for the number of requests per minute is reached. The request size remains limited at 10000 characters. An hourly and a daily quota limit the number of characters that the Microsoft Translator API will accept. |
|                              | AppId is over the quota.                                                                    | The application ID exceeded the hourly or daily quota.                                                                                                                                                                      |

## NOTE

The quota will adjust to ensure fairness among all users of the service.

## Example

### C# PHP

#### C#

```
using System;
using System.IO;
using System.Net;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.Text;
using System.Web;

namespace CtfReporting
{
    class Program
    {
        static void Main(string[] args)
        {
            AdmAccessToken admToken;
            string headerValue;
            //Get Client Id and Client Secret from https://datamarket.azure.com/developer/applications/
            //Refer obtaining AccessToken (http://msdn.microsoft.com/en-us/library/hh454950.aspx)
            AdmAuthentication admAuth = new AdmAuthentication("clientId", "clientSecret");
            try
            {
                admToken = admAuth.GetAccessToken();
                // Create a header with the access_token property of the returned token
                headerValue = "Bearer " + admToken.access_token;
                Console.WriteLine("User translations count");
                ctf GetUserTranslationsCount(headerValue, 0, 5);
                Console.WriteLine("Press Enter for Next 5 Records");
                if (Console.ReadKey().Key == ConsoleKey.Enter)
                {
                    ctf GetUserTranslationsCount(headerValue, 5, 5);
                    Console.WriteLine("Press any key to exit...");
                    Console.ReadKey(true);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("Press any key to exit...");
                Console.ReadKey(true);
            }
        }

        private static void ctf GetUserTranslationsCount(string authToken, int skip, int take)
        {
            // Add CtfReportingService as a service reference,
            Address:http://api.microsofttranslator.com/v2/beta/ctfreporting.svc
            CtfReportingService.CtfReportingServiceClient reportingClient = new
            CtfReportingService.CtfReportingServiceClient();
            CtfReportingService.UserTranslationCount[] userTranslationsCount =
            reportingClient.GetUserTranslationCounts(authToken, "", "es", "en", null, null, "", "general", null, null,
            skip, take);
            foreach (CtfReportingService.UserTranslationCount item in userTranslationsCount)
            {
                Console.WriteLine(string.Format("Count={0},From={1},To={2},Rating={3},Uri={4},User={5}",
                item.Count, item.From, item.To, item.Rating, item.Uri, item.User));
            }
        }
    }
}
```

```

}

[DataContract]
public class AdmAccessToken
{
    [DataMember]
    public string access_token { get; set; }

    [DataMember]
    public string token_type { get; set; }

    [DataMember]
    public string expires_in { get; set; }

    [DataMember]
    public string scope { get; set; }
}

public class AdmAuthentication
{
    public static readonly string DatamarketAccessUri =
"https://datamarket.accesscontrol.windows.net/v2/OAuth2-13";
    private string clientId;
    private string clientSecret;
    private string request;

    public AdmAuthentication(string clientId, string clientSecret)
    {
        this.clientId = clientId;
        this.clientSecret = clientSecret;
        //If clientid or client secret has special characters, encode before sending request
        this.request = string.Format("grant_type=client_credentials&client_id={0}&client_secret={1}&scope=http://api.microsofttranslator.com", HttpUtility.UrlEncode(clientId),
HttpUtility.UrlEncode(clientSecret));
    }

    public AdmAccessToken GetAccessToken()
    {
        return HttpPost(DatamarketAccessUri, this.request);
    }

    private AdmAccessToken HttpPost(string DatamarketAccessUri, string requestDetails)
    {
        //Prepare OAuth request
        WebRequest webRequest = WebRequest.Create(DatamarketAccessUri);
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.Method = "POST";
        byte[] bytes = Encoding.ASCII.GetBytes(requestDetails);
        webRequest.ContentLength = bytes.Length;
        using (Stream outputStream = webRequest.GetRequestStream())
        {
            outputStream.Write(bytes, 0, bytes.Length);
        }
        using (WebResponse webResponse = webRequest.GetResponse())
        {
            DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(AdmAccessToken));
            //Get deserialized object from JSON stream
            AdmAccessToken token = (AdmAccessToken)serializer.ReadObject(webResponse.GetResponseStream());
            return token;
        }
    }
}
}

**PHP**

<?php

class AccessTokenAuthentication {
/*
 * Get the access token.
 *
 * @param string $grantType      Grant type.
 */

```

```

        * @param string $scopeUrl      Application Scope URL.
        * @param string $clientID      Application client ID.
        * @param string $clientSecret Application client ID.
        * @param string $authUrl       Oauth Url.
        *
        * @return string.
        */
    function getTokens($grantType, $scopeUrl, $clientID, $clientSecret, $authUrl){
        try {
            //Initialize the Curl Session.
            $ch = curl_init();
            //Create the request Array.
            $paramArr = array (
                'grant_type'      => $grantType,
                'scope'           => $scopeUrl,
                'client_id'       => $clientID,
                'client_secret'   => $clientSecret
            );
            //Create an Http Query.//
            $paramArr = http_build_query($paramArr);
            //Set the Curl URL.
            curl_setopt($ch, CURLOPT_URL, $authUrl);
            //Set HTTP POST Request.
            curl_setopt($ch, CURLOPT_POST, TRUE);
            //Set data to POST in HTTP "POST" Operation.
            curl_setopt($ch, CURLOPT_POSTFIELDS, $paramArr);
            //CURLOPT_RETURNTRANSFER- TRUE to return the transfer as a string of the return value of
            curl_exec().
            curl_setopt ($ch, CURLOPT_RETURNTRANSFER, TRUE);
            //CURLOPT_SSL_VERIFYPeer- Set FALSE to stop cURL from verifying the peer's certificate.
            curl_setopt($ch, CURLOPT_SSL_VERIFYPeer, false);
            //Execute the cURL session.
            $strResponse = curl_exec($ch);
            //Get the Error Code returned by Curl.
            $curl_errno = curl_errno($ch);
            if($curl_errno){
                $curlError = curl_error($ch);
                throw new Exception($curlError);
            }
            //Close the Curl Session.
            curl_close($ch);
            //Decode the returned JSON string.
            $objResponse = json_decode($strResponse);

            if ($objResponse->error){
                throw new Exception($objResponse->error_description);
            }
            return $objResponse->access_token;
        } catch (Exception $e) {
            echo "Exception-".$e->getMessage();
        }
    }

}

/*
 * Class:AccessTokenAuthentication
 *
 * Create SOAP Object.
 */
class SOAPMicrosoftTranslator {
    /*
     * Soap Object.
     *
     * @var ObjectArray.
     */
    public $objSoap;
    /*
     * Create the SAOP object.
     *

```

```

* @param string $accessToken Access Token string.
* @param string $wsdlUrl      WSDL string.
*
* @return string.
*/
public function __construct($accessToken, $wsdlUrl){
    try {
        //Authorization header string.
        $authHeader = "Authorization: Bearer ". $accessToken;
        $contextArr = array(
            'http'  => array(
                'header' => $authHeader
            )
        );
        //Create a streams context.
        $objContext = stream_context_create($contextArr);
        $optionsArr = array (
            'soap_version' => 'SOAP_1_2',
            'encoding'       => 'UTF-8',
            'exceptions'    => true,
            'trace'          => true,
            'cache_wsdl'     => 'WSDL_CACHE_NONE',
            'stream_context' => $objContext,
            'user_agent'     => 'PHP-SOAP/'.PHP_VERSION."\r\n".$authHeader
        );
        //Call Soap Client.
        $this->objSoap = new SoapClient($wsdlUrl, $optionsArr);
    } catch(Exception $e){
        echo "<h2>Exception Error!</h2>";
        echo $e->getMessage();
    }
}

try {
    //Soap WSDL Url.
    $wsdlUrl      = "http://api.microsofttranslator.com/v2/beta/ctfreporting.svc";
    //Client ID of the application.
    $clientID      = "clientId";
    //Client Secret key of the application.
    $clientSecret  = "clientSecret";
    //OAuth Url.
    $authUrl       = "https://datamarket.accesscontrol.windows.net/v2/OAuth2-13/";
    //Application Scope Url
    $scopeUrl      = "http://api.microsofttranslator.com";
    //Application grant type
    $grantType      = "client_credentials";

    //Create the Authentication object
    $authObj       = new AccessTokenAuthentication();
    //Get the Access token
    $accessToken   = $authObj->getTokens($grantType, $scopeUrl, $clientID, $clientSecret, $authUrl);
    //Create soap translator Object
    $soapTranslator = new SOAPMicrosoftTranslator($accessToken, $wsdlUrl);

    //Set the Params.
    $from          = 'en';
    $to            = 'de';
    $minRating    = 0;
    $maxRating    = 10;
    $takeResult   = 100;
    $user          = 'guestUser';
    //Request argument list.
    $requestArg = array (
        'user'        => $user,
        'from'        => $from,
        'to'          => $to,
        'minRating'  => $minRating,
        'maxRating'  => $maxRating,
    );
}

```

```
        'take'      => $takeResult
    );
//Call GetUserTranslationCount Method.
$responseObj = $soapTranslator->objSoap->GetUserTranslationCounts($requestArg);
$translationCountArr = $responseObj-> GetUserTranslationCountsResult->UserTranslationCount;
echo "<table border=2px>";
echo "<tr>";
echo "<td><b>User</b></td><td><b>From</b></td><td><b>To</b></td><td><b>rating</b></td><td><b>Count</b></td>";
</td>";
echo "</tr>";
if(sizeof($translationCountArr) > 0) {
    if(sizeof($translationCountArr) > 1) {
        foreach ($translationCountArr as $translationCount) {
            echo "<tr><td>$translationCount->User</td><td>$translationCount->From</td>
                <td>$translationCount->To</td><td>$translationCount->Rating</td>
                <td>$translationCount->Count</td></tr>";
        }
    } else {
        echo "<tr><td>$translationCountArr->User</td><td>$translationCountArr->From</td>
            <td>$translationCountArr->To</td><td>$translationCountArr->Rating</td>
            <td>$translationCountArr->Count</td></tr>";
    }
} else {
    echo "<tr><td col='5'>No Record Found.</td></tr>";
}
echo "</table>";
exit;
} catch (Exception $e) {
    echo "Exception: " . $e->getMessage() . PHP_EOL;
}
```

# Speech Translation API

4/12/2017 • 1 min to read • [Edit Online](#)

This service offers a streaming API to transcribe conversational speech from one language into text of another language. The API also integrates text-to-speech capabilities to speak the translated text back. The Speech Translation API enables scenarios like real-time translation of conversations as seen in [Skype Translator](#).

With **Microsoft Translator Speech Translation API**, client applications stream speech audio to the service and receive back a stream of text-based results, which include the recognized text in the source language, and its translation in the target language. Text results are produced by applying Automatic Speech Recognition (ASR) powered by deep neural networks to the incoming audio stream. Raw ASR output is further improved by a new technique called TrueText in order to more closely reflect user intent. For example, TrueText removes disfluencies (the hmms and coughs) and restore proper punctuation and capitalization. The ability to mask or exclude profanities is also included. The recognition and translation engines are specifically trained to handle conversational speech. The Speech Translation service uses silence detection to determine the end of an utterance. After a pause in voice activity, the service will stream back a final result for the completed utterance. The service can also send back partial results, which give intermediate recognitions and translations for an utterance in progress. For final results, the service provides the ability to synthesize speech (text-to-speech) from the spoken text in the target languages. Text-to-speech audio is created in the format specified by the client. WAV and MP3 formats are available.

The Speech Translation API leverages the WebSocket protocol to provide a full-duplex communication channel between the client and the server.

Code samples demonstrating use of the Speech Translation API are available from the [Microsoft Translator Github site](#).

# Microsoft Translator Text Translation

4/12/2017 • 1 min to read • [Edit Online](#)

The Microsoft Translator API seamlessly integrates with your site or app to translate content. It also offers the unique ability to improve the accuracy of the delivered translations through the use of the Collaborative Translations Framework (CTF), allowing users to recommend alternative translations to those provided by Translator's automatic translation engine. It can also be customized with your pre-translated data through the [Microsoft Translator Hub](#). The Translator Web Widget is an easy to integrate one-stop-solution to add automatic translation support to your webpage, blog, or SharePoint site. This free widget will allow you to add this support without the need for any particular coding.

Code samples demonstrating use of the Speech Translation API are available from the [Microsoft Translator Github site](#).

# Video API

4/12/2017 • 2 min to read • [Edit Online](#)

Welcome to Microsoft Video API. Video API is a cloud-based API that provides advanced algorithms for tracking faces, detecting motion, stabilizing and creating thumbnails from video. This API allows you to build more personalized and intelligent apps by understanding and automatically transforming your video content.

## Face Detection and Tracking

The Face Detection and Tracking video API provides high precision face location detection and tracking that can detect up to 64 human faces in a video. Frontal faces provide the best results, while side faces and small faces (smaller than or equal to 24x24 pixels) are challenging.

Face detection can be done by uploading an entire video file or by specifying the URL of an existing video on the web.

The detected and tracked faces are returned with coordinates (left, top, width, and height) indicating the location of faces in the image in pixels, as well as a face ID number indicating the tracking of that individual. Face ID numbers are prone to reset under circumstances when the frontal face is lost or overlapped in the frame.

For more details about how to use face detection and tracking, refer to the [Video API reference guide](#).

## Motion Detection

The Motion Detection API provides indicators once there are objects in motion in a fixed background video (e.g. a surveillance video). The API allows you to input motion detection zones to define areas in the frame to detect the motion. Motion Detection is trained to reduce false alarms. Current limitations of the algorithms include semi-transparent objects and small objects. Developers have the ability to adjust sensitivity levels to detect smaller objects.

Motion detection can be done by uploading an entire video file or by specifying the URL of an existing video on the web.

The output of this API is in JSON format, consisting of both time and duration of motion detected in the video.

For more details about how to use motion detection, refer to the [Video API reference guide](#).

## Stabilization

The Stabilization API provides automatic video stabilization and smoothing for shaky videos. This API uses many of the same technologies found in [Microsoft Hyperlapse](#). Stabilization is optimized for small camera motions, with or without rolling shutter effects (e.g. holding a static camera, walking with a slow speed).

Stabilization can be done by uploading an entire video file or by specifying the URL of an existing video on the web.

The output of this API is in MP4 video format, consisting of the smoothed and stabilized version of the originally submitted video.

For more details about how to use stabilization, refer to the [Video API reference guide](#).

## Video Thumbnail

A video thumbnail lets people see a preview or snapshot of your video. When a viewer wants to get a quick glance

of the video content, this API can be used to generate a motion thumbnail which consists of scenes from the original video.

Microsoft Video API applies computer vision intelligence to identify the characteristics of your video's content. It will select the most representative scenes from your video to create a thumbnail. Selection criteria is also based on video quality, diversity, and stability of the footage. Video API creates an index of the best video scenes and marks the duration in seconds. Based on this index, a user may opt to select a set of scenes to create the thumbnail. The other option is to let Video API generate the thumbnail based on its algorithms.

For more details about how to use video thumbnail, refer to the [Video API reference guide](#).

# Get Started with Video API in C#

4/12/2017 • 3 min to read • [Edit Online](#)

Explore a basic Windows application that uses Microsoft Cognitive Services (formerly Project Oxford) Video API to stabilize videos, recognize motion tracking and detecting faces in a video. The below example lets you submit a locally stored file in mp4, mov, or wmv formats. You can use this open source example as a template for building your own app for Windows using the Video 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 Video API and get a subscription key

Before creating the example, you must subscribe to Video API which is part of Microsoft Cognitive services. For subscription and key management details, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial. Make sure to follow best practices for keeping your API key secret and secure.

### Get the client library and example

You may download the Video API client library and example via [SDK](#). The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder.

## Step 1: Install the example

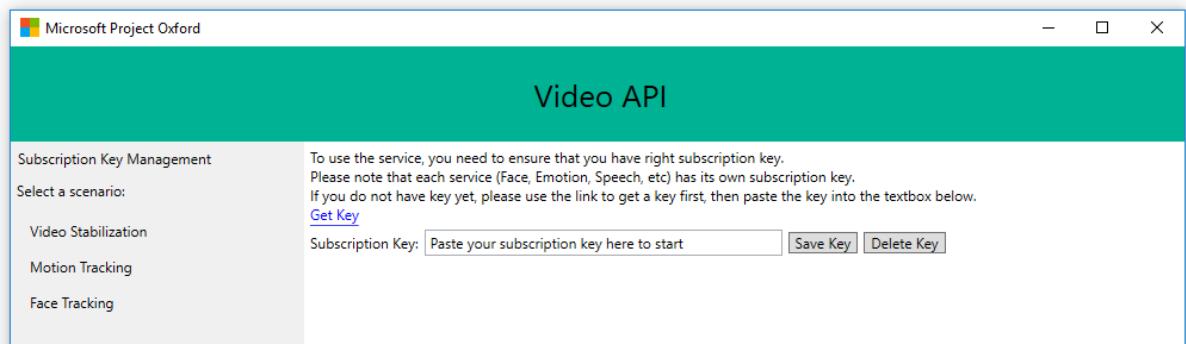
1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Browse to the folder where you saved the downloaded Video API files. Click on **Video**, then **Windows**, and then the **Sample-WPF** folder.
3. Double-click to open the Visual Studio 2015 Solution (.sln) file named **VideoAPI-WPF-Samples.sln**. This will open the solution in Visual Studio.

## Step 2: Build the example

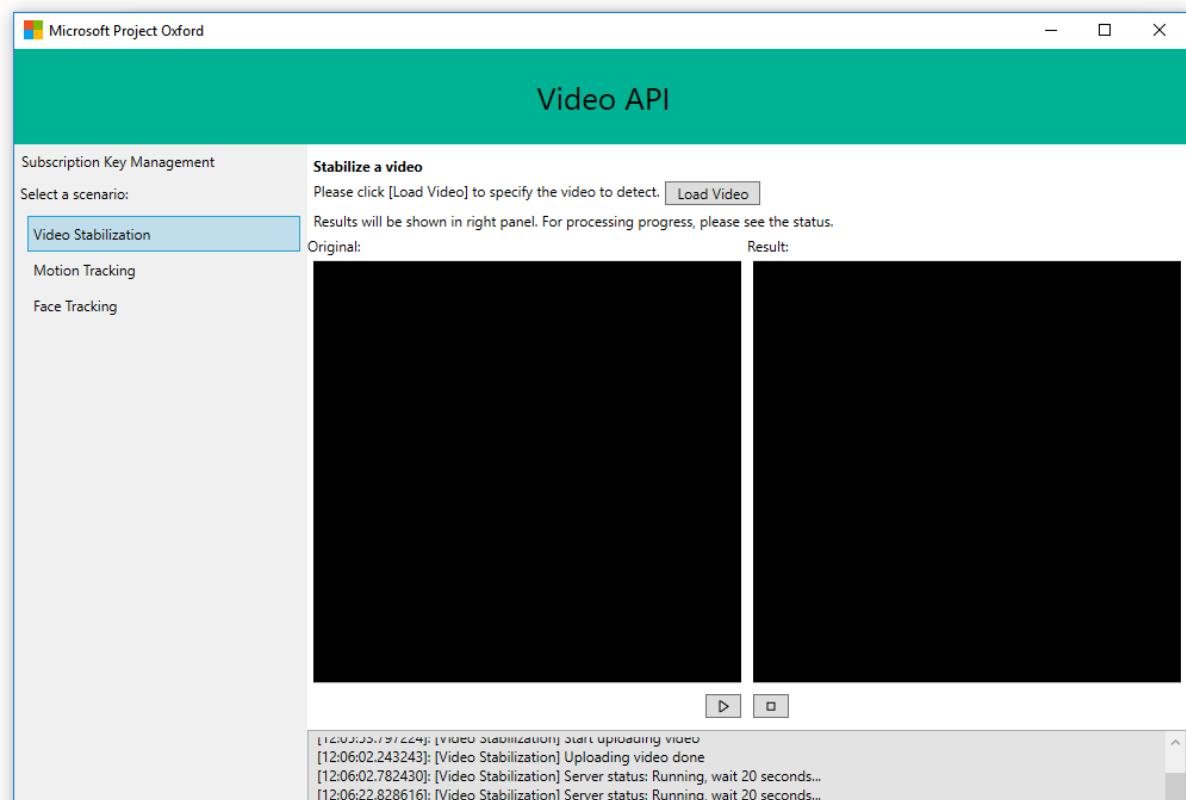
1. In **Solution Explorer** you will find that the solution consists of three projects, **SampleUserControlLibrary**, **VideoSDK (Portable)** and **VideoAPI-WPF-Samples**. Right-click **VideoAPI-WPF\_Samples** and in the pop-up menu locate and select "**Set as StartUp Project**".
2. Press Ctrl+Shift+B, or click **Build** on the ribbon menu, then select **Build Solution**.

## Step 3: Run the example

1. After the build is complete, press **F5** or click **Start** on the ribbon menu to run the example.
2. Locate the window with the **text edit box** reading "**Paste your subscription key here to start**" as shown in below screenshot. Paste your subscription key into the text box. 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.



- Under "Select Scenario" click to use either of the three scenarios, "**Video Stabilization**", "**Motion Tracking**" or "**Face Detection**".



- Click the "**Load Video**" button to browse to a video file stored on your PC, laptop or local network. Video must be in mp4, mov, or wmv format. Expect to wait a minute or two for the file to load.

- Video stabilization:** Click the rectangular start button to run your video. Your original video will run in the left window and a stabilized version in the right window.
- Motion Tracking:** Original video will run in the left window. If motion is detected, a red rectangle will frame your video. In the right window, Json will be displayed indicating the region (at this point there is only one, the entire frame) and duration of movements.
- Face Detection:** Original video will run in the left window. If or when faces are detected, Json coordinates (left, top, width, and height) are returned in the right window indicating the location of faces in the video in pixels, as well as a face ID indicating the tracking number of the person. If no human faces are detected, it will be indicated as "FacesDetected": null.
- Lower horizontal window:** Progress during loading and recognition will be displayed in real time.

- Microsoft receives the videos you upload and may use them to improve Video API and related services. By submitting a video, you confirm that you have followed our [Developer Code of Conduct](#).

## Review summary

```
public MainWindow()
{
    InitializeComponent();

    //
    // Initialize SampleScenarios User Control with titles and scenario pages
    //
    _scenariosControl.SampleTitle = "Video API";
    _scenariosControl.SampleScenarioList = new Scenario[]
    {
        new Scenario { Title = "Video Stabilization", PageClass = typeof(StabilizationPage) },
        new Scenario { Title = "Motion Tracking", PageClass = typeof(MotionDetectionPage) },
        new Scenario { Title = "Face Tracking", PageClass = typeof(FaceTrackingPage) },
    };
}
```

Code snippets with suggestions on how to extend and and customize your video app, coming in the near future.  
Check back soon!

# How to Call Video APIs

4/12/2017 • 8 min to read • [Edit Online](#)

The samples are written in C# using the Video API client library.

## Concepts

If you are not familiar with any of the following concepts in this guide, please refer to the definitions in our [glossary](#) at any time.

- Stabilization
- Face Detection and Tracking
- Motion Detection

## Preparation

In the below examples the following features are demonstrated:

- Creating a stabilized video from a shaky video
- Analyzing faces detected in a video and outputting the corresponding JSON
- Analyzing motion detected in a video and outputting the corresponding JSON

In order to implement these features, you will need a video with the below specified characteristics for each of the features you want to try.

- For stabilization: A video that is shaky or jittery.
- For face detection and tracking: A video where people's faces are facing the camera.
- For motion detection: A video with a stationary background and some movement in the foreground.

## Step 1: Authorize the API call

Every call to the Video API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header. To pass the subscription key through a query string, refer to the Video API request URL below as an example:

```
https://westus.api.cognitive.microsoft.com/video/v1.0/stabilize&subscription-key=<Your subscription key>
```

As an alternative, the subscription key can also be specified in the HTTP request header:

```
ocp-apim-subscription-key: <Your subscription key>
```

When using a client library, the subscription key is passed in through the constructor of the `VideoServiceClient` class. For example:

```
var videoServiceClient = new VideoServiceClient("Your subscription key");
```

To obtain a subscription key, see [Subscriptions](#).

## Step 2: Upload a video to the service and check the status

The most basic way to perform any of the Video API calls is by uploading a video directly. This is done by sending a "POST" request with application/octet-stream content type together with the data read from a video file. The maximum size of the video is 100MB.

Using the client library, stabilization by means of uploading is done by passing in a stream object. See the example below:

```
Operation videoOperation;
using (var fs = new FileStream(@"C:\Videos\Sample.mp4", FileMode.Open))
{
    videoOperation = await videoServiceClient.CreateOperationAsync(fs, OperationType.Stabilize);
}
```

*Please note that the CreateOperationAsync method of VideoServiceClient is async. The calling method should be marked as async as well in order to use the await clause.*

If the video is already on the web and has a public URL, Video APIs can be accessed by providing the URL. In this example, the request body will be a JSON string which contains the URL.

Using the client library, stabilization by means of an URL can easily be executed using another overload of the CreateOperationAsync method.

```
var videoUrl = "http://www.example.com/sample.mp4";
Operation videoOperation = await videoServiceClient.CreateOperationAsync(videoUrl, OperationType.Stabilize);
```

This upload method will be the same for all the Video API calls. The only difference will be the POST calls to the various APIs you want to execute (POST stabilize, trackFace, detectMotion).

Once you have uploaded a video to the Video API call you want to execute, the next operation you will want to perform is to check its status. Because video files are usually larger and more diverse than other files, users can expect a long processing time at this step. The time depends on the size and the length of the file.

Using the client library, you can retrieve the operation status and result using the GetOperationResultAsync method.

```
var operationResult = await videoServiceClient.GetOperationResultAsync(videoOperation);
```

Typically, the client side should periodically retrieve the operation status until the status is shown as "Succeeded" or "Failed".

```

OperationResult operationResult;
while (true)
{
    operationResult = await videoServiceClient.GetOperationResultAsync(videoOperation);
    if (operationResult.Status == OperationStatus.Succeeded || operationResult.Status ==
OperationStatus.Failed)
    {
        break;
    }

    Task.Delay(30000).Wait();
}

```

When the status of OperationResult is shown as "Succeeded", depending on the video operation you created, the result can be retrieved in the following ways:

| VIDEO OPERATION                              | HOW TO RETRIEVE RESULTS?                                                                                                            | SAMPLE CODE                                                       |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| Stabilization                                | The result (as a video file) can be retrieved from the URL specified in the ResourceLocation field of the OperationResult instance. | var stabilizationResultUrl = operationResult.ResourceLocation;    |
| Face Detection/Tracking and Motion Detection | The result (as a JSON string) is available in the ProcessingResult field of OperationResult.                                        | var motionDetectionJsonString = operationResult.ProcessingResult; |

## Step 3: Retrieving video output files for stabilization

The output from the Stabilization Video API is stabilized in MP4 video format. Once your video output is ready, as informed by the GetVideoOperationResultAsync method, you can download your video using the GetResultVideoAsync method.

```

var videoStream = await videoServiceClient.GetResultVideoAsync(resultUrl);
using (var fileStream = File.Create(outputFile))
{
    videoStream.CopyTo(fileStream);
}

```

## Step 4: Retrieving and understanding the face detection and tracking JSON output

For the face detection and tracking operation, the output result contains the metadata from the faces within the given file in JSON format.

As explained in Step 2, the JSON output is available in the ProcessingResult field of OperationResult, when its status is shown as "Succeeded".

The face detection and tracking JSON includes the following attributes:

| ATTRIBUTE | DESCRIPTION                                  |
|-----------|----------------------------------------------|
| Version   | Refers to the version of the Video API JSON. |

| ATTRIBUTE     | DESCRIPTION                                                                                                                                                                                                                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Timescale     | "Ticks" per second of the video.                                                                                                                                                                                                                                                                              |
| Offset        | The time offset for timestamps. In version 1.0 of Video APIs, this will always be 0. In future supported scenarios, this value may change.                                                                                                                                                                    |
| Framerate     | Frames per second of the video.                                                                                                                                                                                                                                                                               |
| Fragments     | The metadata is cut up into different smaller pieces called fragments. Each fragment contains a start, duration, interval number, and event(s).                                                                                                                                                               |
| Start         | The start time of the first event, in ticks.                                                                                                                                                                                                                                                                  |
| Duration      | The length of the fragment, in ticks.                                                                                                                                                                                                                                                                         |
| Interval      | The length of each event within the fragment, in ticks.                                                                                                                                                                                                                                                       |
| Events        | An array of events. The outer array represents one interval of time. The inner array consists of 0 or more events that happened at that point in time.                                                                                                                                                        |
| ID            | Index of a face. A given individual should have the same ID throughout the overall video. Due to limitations in the detection algorithm (e.g. occlusion) this cannot be guaranteed.                                                                                                                           |
| X, Y          | The upper left X and Y coordinates of the face bounding box in a normalized scale of 0.0 to 1.0. The X and Y coordinates are relative to the landscape orientation of a video. For example, if you have a portrait video, you will have to transpose the coordinates accordingly .                            |
| Width, Height | The width and height of the face bounding box in a normalized scale of 0.0 to 1.0.                                                                                                                                                                                                                            |
| facesDetected | Attribute is found at the end of the JSON results and summarizes the number of faces that the algorithm detected during the video. Because the IDs can be reset if a face becomes undetected (e.g. face goes off screen, looks away), this number may not always equal the true number of faces in the video. |

The reason for formatting the JSON this way is to set up the APIs for future scenarios, where it will be important to retrieve metadata quickly and manage a large stream of results. This formatting is using both the techniques of fragmentation (allowing you to break up the metadata in time-based portions, where you can download only what you need), and segmentation (allowing you to break up the events if they get too large). Some simple calculations can help you transform the data. For example, if an event started at 6300 (ticks), with a timescale of 2997 (ticks/sec) and a framerate of 29.97 (frames/sec), then:

- Start/Timescale = 2.1 seconds
- Seconds x (Framerate/Timescale) = 63 frames

Below is a simple example of extracting the JSON into a per frame format for face detection and tracking:

```

var faceDetectionTrackingResultJsonString = operationResult.ProcessingResult;
var faceDetectionTracking = JsonConvert.DeserializeObject<FaceDetectionResult>
(faceDetectionTrackingResultJsonString, settings);

```

## Step 5: Retrieving and understanding the motion detection JSON output

You retrieve the motion detection JSON results in exactly the same way as with the face detection and tracking method. Please see Step 4 to learn how to retrieve the results.

The motion detection JSON has similar concepts as the face detection and tracking JSON. Because the motion detection JSON only needs to record when motion happened and the duration of motion, there are a few differences.

| ATTRIBUTE     | DESCRIPTION                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Version       | Refers to the version of the Video API JSON.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Timescale     | "Ticks" per second of the video.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Offset        | The time offset for timestamps. In version 1.0 of Video APIs, this will always be 0. In future supported scenarios, this value may change.                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Framerate     | Frames per second of the video.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Fragments     | The metadata is cut up into smaller pieces called fragments. Each fragment contains a start, duration, interval number, and event(s). A fragment with no events means that no motion was detected during that start time and duration.                                                                                                                                                                                                                                                                                                                                                      |
| Width, Height | Refers to the width and height of the video in pixels.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Regions       | Regions refer to the area(s) in your video where you care about motion. <b>1. ID</b> represents the region area. <b>2. TYPE</b> represents the shape of the region you are concerned about in regards to motion. In this version, it is always a polygon. <b>3. The region</b> is defined by an array of polygon points, each point has X and Y coordinates in a normalized scale of 0.0 to 1.0. <b>4. The X and Y coordinates</b> are relative to the landscape orientation of a video. For example, if you have a portrait video, you will have to transpose the coordinates accordingly. |
| Start         | The start time of the first event, in ticks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Duration      | The length of the fragment, in ticks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Interval      | The length of each event within the fragment, in ticks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Event         | An array of events. The outer array represents one interval of time. The inner array consists of 0 or more events that happened at that point in time.                                                                                                                                                                                                                                                                                                                                                                                                                                      |

| ATTRIBUTE | DESCRIPTION                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type      | In the current version, the integer 2 refers to when motion happened, integer 4 refers to when light changes happen.                                                                                                                                                                                                                                                                                     |
| TypeName  | Will be "motion" when Type is 2 and "light change" when Type is 4. (This is a string).                                                                                                                                                                                                                                                                                                                   |
| Location  | An array of motion locations indicating where motion happens. There will always be one location in this version. The motion location has dimensions in X, Y, Width, and Height. The X and Y coordinates represent the upper left hand X, Y coordinates of the location in a normalized scale of 0.0 to 1.0. The width and height represent the size of the location in a normalized scale of 0.0 to 1.0. |

Below is a simple example of extracting the JSON into a per frame motion detection result:

```
var motionDetectionJsonString = operationResult.ProcessingResult;
var motionDetection = JsonConvert.DeserializeObject<MotionDetectionResult>(motionDetectionJsonString,
settings);
```

## Summary

You have now learned about the functionalities of the Video API: how you can upload a video, check its status, retrieve a stabilized video, and retrieve face detection/tracking and motion detection metadata.

For more information about API details, see the API reference guide "[Video API Reference](#)".

# How to Analyze Videos in Real-time

4/12/2017 • 7 min 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 uses 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();

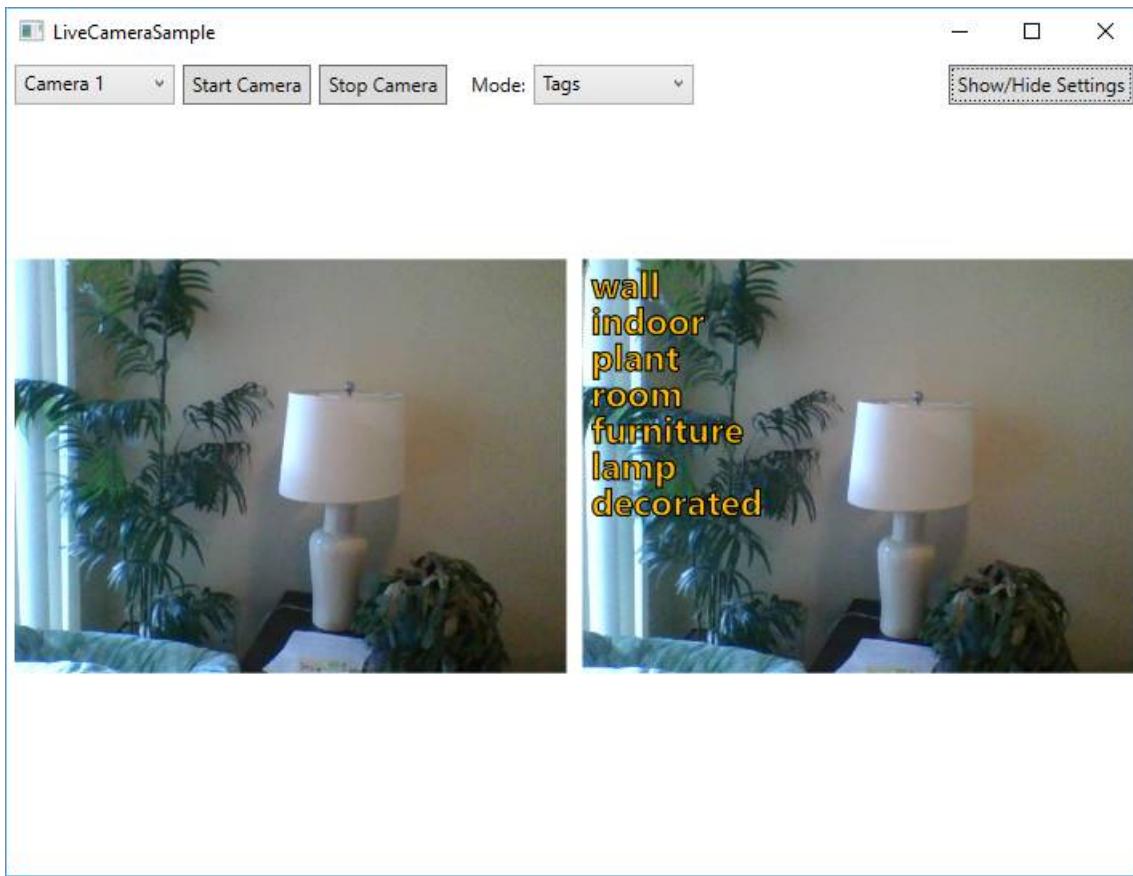
            // Wait for keypress to stop
            Console.WriteLine("Press any key to stop...");
            Console.ReadKey();

            // 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 [microsoft.com/cognitive](https://microsoft.com/cognitive). 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.

To report abuse of the Microsoft Cognitive Services to Microsoft, please visit the [Microsoft Cognitive Services website](#), and use the "Report Abuse" link at the bottom of the page to contact Microsoft. For more information about Microsoft privacy policies please see their privacy statement here: <https://go.microsoft.com/fwlink/?LinkId=521839>.

## 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](#).

# Glossary

4/12/2017 • 2 min to read • [Edit Online](#)

A

B

C

D

## **Duration**

The length of the fragment, in ticks (within the JSON result).

E

## **Events**

An array of events (within the JSON result). The outer array represents one interval of time. The inner array consists of 0 or more events that happened at that point in time.

F

## **Face Detection/Tracking**

This is derived from the video face detection and tracking results (this is different from the Face API's Face ID). This is the index of a face. A given individual should have the same ID throughout the overall video. Due to limitations in the detection algorithm (e.g. occlusion) this cannot be guaranteed.

## **Fragments**

The JSON metadata is chunked up into different segments called fragments. Each fragment contains a start, duration, interval number, and event(s).

## **Framerate**

Frames per second of the video.

G

H

I

## **Interval**

The length of each event within the fragment, in ticks (within the JSON result).

J

K

L

M

### **Motion Detection**

Motion detection is the action of detecting motion in a video.

N

O

### **Offset**

This is the time offset for timestamps (within the JSON result). In version 1.0 of Video APIs, this will always be 0. In future scenarios we support, this value may change.

P

Q

R

### **Region of Interest**

This is derived from the motion detection JSON results. Regions refers to the area in your video where you care about motion.

S

### **Sensitivity**

User can specify 'sensitivity level' for motion detection, which are high, medium, and low, default is medium. Higher sensitivity means more motions will be detected at a cost that more false alarms will be reported.

### **Stabilization**

Stabilization takes a shaky video and smooths and stabilizes it.

### **Start**

The start time of the first event, in ticks (within the JSON result).

### **Subscription key**

Subscription key is a string that you need to specify as a query string parameter in order to invoke any Face API. The subscription key can be found in My Subscriptions page after you sign in the Microsoft Cognitive Services portal. There will be two keys associated with each subscription: one primary key and one secondary key. Both can be used to invoke API identically. You need to keep the subscription keys secure, and you can regenerate subscription keys at any time from My Subscriptions page as well.

T

### **Timescale**

The "ticks" per second of the video (within the JSON result).

### **Type**

This is derived from the motion detection JSON results. 2 refers to motion, 4 refers to light change.

U

V

## Video API

Video API is a cloud-based API that provides the most advanced algorithms for stabilization, face detection and tracking, and motion detection in video.

W

X

Y

Z

- [Link back to Overview](#)
- [Link back to Get Started with Video API](#)
- [Link back to How to Call Video API](#)

# Web Language Model API Overview

4/12/2017 • 1 min to read • [Edit Online](#)

Welcome to the Microsoft Web Language Model API, a REST-based cloud service providing state-of-the-art tools for natural language processing. Using this API, your application can leverage the power of big data through language models trained on web-scale corpora collected by Bing in the EN-US market.

These smoothed backoff N-gram language models, supporting Markov order up to 5, are trained on the following corpora:

- Web page body text
- Web page title text
- Web page anchor text
- Web search query text

The Web LM REST API supports four lookup operations:

1. Joint (log10) probability of a sequence of words.
2. Conditional (log10) probability of one word given a sequence of preceding words.
3. List of words (completions) most likely to follow a given sequence of words.
4. Word breaking of strings that contain no spaces.

## Getting Started

1. Subscribe to the service.
2. Download the [SDK](#).
3. Run the SDK sample code.
4. Consult the [API Reference](#) for further details, including code snippets in a variety of languages.

## Underlying Technology

The following paper provides details on the development of these language models, and should be cited in research publications that utilize this service:

- [An Overview of Microsoft Web N-gram Corpus and Applications](#), NAACL-HLT 2010

Click [here](#) for a current list of papers citing this work.