

Contents

[Translator Text Documentation](#)

[Translator Text API](#)

[Overview](#)

[What is the Translator Text API?](#)

[Quickstarts](#)

[Translate text](#)

[Transliterate text](#)

[Detect language](#)

[Look up word translations](#)

[Get supported languages](#)

[Determine sentence length](#)

[Tutorials](#)

[Create a WPF app for Translator Text, C#](#)

[Create a Flask translation app, Python](#)

[Concepts](#)

[Customize and improve text translation](#)

[How the API counts characters](#)

[How-to guides](#)

[Sign-up for Translator Text API](#)

[Migrate to V3 Text API](#)

[Add profanity filtering](#)

[Receive word alignment information](#)

[Prevent translation of content](#)

[Use the dynamic dictionary feature](#)

[Translate behind firewalls](#)

[Samples](#)

[C#](#)

[Java](#)

[Python](#)

[Node.js](#)

[Go](#)

[PHP](#)

[Ruby](#)

Reference

[v3 Translator Text API reference](#)

[Languages](#)

[Translate](#)

[Transliterate](#)

[Detect](#)

[BreakSentence](#)

[Dictionary Lookup](#)

[Dictionary Examples](#)

[v2 Translator Text API reference \(Deprecated\)](#)

[Transform text](#)

[How to use Collaborative Translation Framework \(CTF\) reporting](#)

[Return N-Best translations](#)

Resources

[Code samples on GitHub](#)

[FAQ](#)

[Language and region support](#)

[Pricing](#)

[Regional availability](#)

[Compliance](#)

[Request limits](#)

[Stack Overflow](#)

Custom Translator

Overview

[What is the Custom Translator?](#)

Quickstarts

[Build a custom model](#)

Concepts

[Workspaces and projects](#)

[Parallel documents](#)

[Dictionary](#)

[Document formats](#)

[Sentence alignment](#)

[Data filtering](#)

[Training and model](#)

[BLEU Score](#)

[How-to guides](#)

[Create a project](#)

[Manage projects](#)

[Upload a document](#)

[View document details](#)

[Train a model](#)

[View model details](#)

[View system test results](#)

[Manage settings](#)

[Migrate from Hub](#)

[Unsupported language deployments](#)

[Resources](#)

[Language Support](#)

[Glossary](#)

[FAQ](#)

[Pricing](#)

[Compliance](#)

[Stack Overflow](#)

What is the Translator Text API?

12/10/2019 • 2 minutes to read • [Edit Online](#)

The Translator Text API is easy to integrate in your applications, websites, tools, and solutions. It allows you to add multi-language user experiences in [more than 60 languages](#), and can be used on any hardware platform with any operating system for text-to-text language translation.

The Translator Text API is part of the Azure [Cognitive Services API](#) collection of machine learning and AI algorithms in the cloud, and is readily consumable in your development projects.

About Microsoft Translator

Microsoft Translator is a cloud-based machine translation service. The core service is the Translator Text API, which powers a number of Microsoft products and services, and is used by thousands of businesses worldwide in their applications and workflows, which allows their content to reach a global audience.

Speech translation, powered by the Translator Text API, is also available through the [Microsoft Speech Service](#). It combines functionality from the Translator Speech API and the CustomSpeechService into a unified and fully customizable service. Speech Service is replacing the Translator Speech API, which will be decommissioned on October 15, 2019.

Language support

Microsoft Translator provides multi-language support for translation, transliteration, language detection, and dictionaries. See [language support](#) for a complete list, or access the list programmatically with the [REST API](#).

Microsoft Translator Neural Machine Translation

Neural Machine Translation (NMT) is the new standard for high-quality AI-powered machine translations. It replaces the legacy Statistical Machine Translation (SMT) technology that reached a quality plateau in the mid-2010s.

NMT provides better translations than SMT not only from a raw translation quality scoring standpoint but also because they will sound more fluent and human. The key reason for this fluidity is that NMT uses the full context of a sentence to translate words. SMT only took the immediate context of a few words before and after each word.

NMT models are at the core of the API and are not visible to end users. The only noticeable difference is improved translation quality, especially for languages such as Chinese, Japanese, and Arabic.

Learn more about [how NMT works](#)

Language customization

An extension of the core Microsoft Translator service, Custom Translator can be used in conjunction with the Translator Text API to help you customize the neural translation system and improve the translation for your specific terminology and style.

With Custom Translator, you can build translation systems that handle the terminology used in your own business or industry. Your customized translation system will then easily integrate into your existing applications, workflows, and websites, across multiple types of devices, through the regular Microsoft Translator Text API, by using the category parameter.

Learn more about [language customization](#)

Next steps

- [Sign up](#) for an access key.
- [API reference](#) provides the technical documentation for the APIs.
- [Pricing details](#)

Quickstart: Use the Translator Text API to translate text

12/10/2019 • 21 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to translate a text string from English to German, Italian, Japanese, and Thai using the Translator Text REST API.

This quickstart requires an [Azure Cognitive Services account](#) with a Translator Text resource. If you don't have an account, you can use the [free trial](#) to get a subscription key.

Prerequisites

This quickstart requires:

- C# 7.1 or later
- [.NET SDK](#)
- [Json.NET NuGet Package](#)
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a .NET Core project

Open a new command prompt (or terminal session) and run these commands:

```
dotnet new console -o translate-sample
cd translate-sample
```

The first command does two things. It creates a new .NET console application, and creates a directory named `translate-sample`. The second command changes to the directory for your project.

Next, you'll need to install Json.Net. From your project's directory, run:

```
dotnet add package Newtonsoft.Json --version 11.0.2
```

Select the C# language version

This quickstart requires C# 7.1 or later. There are a few ways to change the C# version for your project. In this guide, we'll show you how to adjust the `translate-sample.csproj` file. For all available options, such as changing the language in Visual Studio, see [Select the C# language version](#).

Open your project, then open `translate-sample.csproj`. Make sure that `LangVersion` is set to 7.1 or later. If there isn't a property group for the language version, add these lines:

```
<PropertyGroup>
  <LangVersion>7.1</LangVersion>
</PropertyGroup>
```

Add required namespaces to your project

The `dotnet new console` command that you ran earlier created a project, including `Program.cs`. This file is where you'll put your application code. Open `Program.cs`, and replace the existing using statements. These statements ensure that you have access to all the types required to build and run the sample app.

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
// Install Newtonsoft.Json with NuGet
using Newtonsoft.Json;
```

Create classes for the JSON response

Next, we're going to create a set of classes that are used when deserializing the JSON response returned by the Translator Text API.

```

/// <summary>
/// The C# classes that represents the JSON returned by the Translator Text API.
/// </summary>
public class TranslationResult
{
    public DetectedLanguage DetectedLanguage { get; set; }
    public TextResult SourceText { get; set; }
    public Translation[] Translations { get; set; }
}

public class DetectedLanguage
{
    public string Language { get; set; }
    public float Score { get; set; }
}

public class TextResult
{
    public string Text { get; set; }
    public string Script { get; set; }
}

public class Translation
{
    public string Text { get; set; }
    public TextResult Transliteration { get; set; }
    public string To { get; set; }
    public Alignment Alignment { get; set; }
    public SentenceLength SentLen { get; set; }
}

public class Alignment
{
    public string Proj { get; set; }
}

public class SentenceLength
{
    public int[] SrcSentLen { get; set; }
    public int[] TransSentLen { get; set; }
}

```

Get subscription information from environment variables

Add the following lines to the `Program` class. These lines read your subscription key and endpoint from environment variables, and throws an error if you run into any issues.


```
private const string key_var = "TRANSLATOR_TEXT_SUBSCRIPTION_KEY";
private static readonly string subscriptionKey = Environment.GetEnvironmentVariable(key_var);

private const string endpoint_var = "TRANSLATOR_TEXT_ENDPOINT";
private static readonly string endpoint = Environment.GetEnvironmentVariable(endpoint_var);

static Program()
{
    if (null == subscriptionKey)
    {
        throw new Exception("Please set/export the environment variable: " + key_var);
    }
    if (null == endpoint)
    {
        throw new Exception("Please set/export the environment variable: " + endpoint_var);
    }
}
// The code in the next section goes here.
```

Create a function to translate text

In the `Program` class, create an asynchronous function called `TranslateTextRequest()`. This function takes four arguments: `subscriptionKey`, `host`, `route`, and `inputText`.

```
// This sample requires C# 7.1 or later for async/await.
// Async call to the Translator Text API
static public async Task TranslateTextRequest(string subscriptionKey, string endpoint, string route, string
inputText)
{
    /*
     * The code for your call to the translation service will be added to this
     * function in the next few sections.
     */
}
```

Serialize the translation request

Next, we need to create and serialize the JSON object that includes the text you want to translate. Keep in mind, you can pass more than one object in the `body`.

```
object[] body = new object[] { new { Text = inputText } };
var requestBody = JsonConvert.SerializeObject(body);
```

Instantiate the client and make a request

These lines instantiate the `HttpClient` and the `HttpRequestMessage`:

```
using (var client = new HttpClient())
using (var request = new HttpRequestMessage())
{
    // In the next few sections you'll add code to construct the request.
}
```

Construct the request and print the response

Inside the `HttpRequestMessage` you'll:

- Declare the HTTP method
- Construct the request URI
- Insert the request body (serialized JSON object)
- Add required headers
- Make an asynchronous request
- Print the response using the classes you created earlier

Add this code to the `HttpRequestMessage`:

```
// Build the request.
// Set the method to Post.
request.Method = HttpMethod.Post;
// Construct the URI and add headers.
request.RequestUri = new Uri(endpoint + route);
request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");
request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

// Send the request and get response.
HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);
// Read response as a string.
string result = await response.Content.ReadAsStringAsync();
// Deserialize the response using the classes created earlier.
TranslationResult[] deserializedOutput = JsonConvert.DeserializeObject<TranslationResult[]>(result);
// Iterate over the deserialized results.
foreach (TranslationResult o in deserializedOutput)
{
    // Print the detected input language and confidence score.
    Console.WriteLine("Detected input language: {0}\nConfidence score: {1}\n", o.DetectedLanguage.Language,
o.DetectedLanguage.Score);
    // Iterate over the results and print each translation.
    foreach (Translation t in o.Translations)
    {
        Console.WriteLine("Translated to {0}: {1}", t.To, t.Text);
    }
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Put it all together

The last step is to call `TranslateTextRequest()` in the `Main` function. In this sample, we're translating to German (`de`), Italian (`it`), Japanese (`ja`), and Thai (`th`). Locate `static void Main(string[] args)` and replace it with this code:

```
static async Task Main(string[] args)
{
    // This is our main function.
    // Output languages are defined in the route.
    // For a complete list of options, see API reference.
    // https://docs.microsoft.com/azure/cognitive-services/translator/reference/v3-0-translate
    string route = "/translate?api-version=3.0&to=de&to=it&to=ja&to=th";
    // Prompts you for text to translate. If you'd prefer, you can
    // provide a string as textToTranslate.
    Console.Write("Type the phrase you'd like to translate? ");
    string textToTranslate = Console.ReadLine();
    await TranslateTextRequest(subscriptionKey, endpoint, route, textToTranslate);
    Console.WriteLine("Press any key to continue.");
    Console.ReadKey();
}
```

You'll notice that in `Main`, you're declaring `subscriptionKey`, `endpoint`, and `route`. Additionally, you're prompting the user for input with `Console.ReadLine()` and assigning the value to `textToTranslate`.

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to your project directory and run:

```
dotnet run
```

Sample response

After you run the sample, you should see the following printed to terminal:

```
Detected input language: en
Confidence score: 1

Translated to de: Hallo Welt!
Translated to it: Salve, mondo!
Translated to ja: ハローワールド!
Translated to th: สวัสดีชาวโลก!
```

This message is built from the raw JSON, which will look like this:

```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "translations": [
      {
        "text": "Hallo Welt!",
        "to": "de"
      },
      {
        "text": "Salve, mondo!",
        "to": "it"
      },
      {
        "text": "ハローワールド!",
        "to": "ja"
      },
      {
        "text": "สวัสดีชาวโลก!",
        "to": "th"
      }
    ]
  }
]
```

Clean up resources

Make sure to remove any confidential information from your sample app's source code, like subscription keys.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [JDK 7 or later](#)
- [Gradle](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Initialize a project with Gradle

Let's start by creating a working directory for this project. From the command line (or terminal), run this command:

```
mkdir translator-sample
cd translator-sample
```

Next, you're going to initialize a Gradle project. This command will create essential build files for Gradle, most importantly, the `build.gradle.kts`, which is used at runtime to create and configure your application. Run this command from your working directory:

```
gradle init --type basic
```

When prompted to choose a **DSL**, select **Kotlin**.

Configure the build file

Locate `build.gradle.kts` and open it with your favorite IDE or text editor. Then copy in this build configuration:

```
plugins {
    java
    application
}
application {
    mainClassName = "Translate"
}
repositories {
    mavenCentral()
}
dependencies {
    compile("com.squareup.okhttp:okhttp:2.5.0")
    compile("com.google.code.gson:gson:2.8.5")
}
```

Take note that this sample has dependencies on OkHttp for HTTP requests, and Gson to handle and parse JSON. If you'd like to learn more about build configurations, see [Creating New Gradle Builds](#).

Create a Java file

Let's create a folder for your sample app. From your working directory, run:

```
mkdir -p src/main/java
```

Next, in this folder, create a file named `Translate.java`.

Import required libraries

Open `Translate.java` and add these import statements:

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
import com.squareup.okhttp.*;
```

Define variables

First, you'll need to create a public class for your project:

```
public class Translate {
    // All project code goes here...
}
```

Add these lines to the `Translate` class. First, the subscription key and endpoint are being read from environment variables. Then, you'll notice that along with the `api-version`, two additional parameters have been appended to the `url`. These parameters are used to set the translation outputs. In this sample, it's set to German (`de`) and Italian (`it`).

```
private static String subscriptionKey = System.getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY");
private static String endpoint = System.getenv("TRANSLATOR_TEXT_ENDPOINT");
String url = endpoint + "/translate?api-version=3.0&to=de,it";
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Create a client and build a request

Add this line to the `Translate` class to instantiate the `OkHttpClient`:

```
// Instantiates the OkHttpClient.
OkHttpClient client = new OkHttpClient();
```

Next, let's build the POST request. Feel free to change the text for translation. The text must be escaped.

```
// This function performs a POST request.
public String Post() throws IOException {
    MediaType mediaType = MediaType.parse("application/json");
    RequestBody body = RequestBody.create(mediaType,
        "[{\n\t\"Text\": \"Welcome to Microsoft Translator. Guess how many languages I speak!\n}]]");
    Request request = new Request.Builder()
        .url(url).post(body)
        .addHeader("Ocp-Apim-Subscription-Key", subscriptionKey)
        .addHeader("Content-type", "application/json").build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

Create a function to parse the response

This simple function parses and prettifies the JSON response from the Translator Text service.

```
// This function prettifies the json response.
public static String prettify(String json_text) {
    JsonParser parser = new JsonParser();
    JsonElement json = parser.parse(json_text);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    return gson.toJson(json);
}
```

Put it all together

The last step is to make a request and get a response. Add these lines to your project:

```
public static void main(String[] args) {
    try {
        Translate translateRequest = new Translate();
        String response = translateRequest.Post();
        System.out.println(prettify(response));
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to the root of your working directory and run:

```
gradle build
```

When the build completes, run:

```
gradle run
```

Sample response

```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "translations": [
      {
        "text": "Willkommen bei Microsoft Translator. Erraten Sie, wie viele Sprachen ich spreche!",
        "to": "de"
      },
      {
        "text": "Benvenuti a Microsoft Translator. Indovinate quante lingue parlo!",
        "to": "it"
      }
    ]
  }
]
```

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Python 2.7.x or 3.x](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Python project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `translate-text.py`. Be sure your IDE's interpreter references the correct version of Python to avoid libraries not being recognized.

```
# -*- coding: utf-8 -*-  
import os, requests, uuid, json
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
pip install requests uuid
```

The first comment tells your Python interpreter to use UTF-8 encoding. Then required modules are imported to read your subscription key from an environment variable, construct the http request, create a unique identifier, and handle the JSON response returned by the Translator Text API.

Set the subscription key, endpoint, and path

This sample will try to read your Translator Text subscription key and endpoint from the environment variables:

`TRANSLATOR_TEXT_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscription_key` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:


```
key_var_name = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY'
if not key_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(key_var_name))
subscription_key = os.environ[key_var_name]

endpoint_var_name = 'TRANSLATOR_TEXT_ENDPOINT'
if not endpoint_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))
endpoint = os.environ[endpoint_var_name]
```

The Translator Text global endpoint is set as the `endpoint`. `path` sets the `translate` route and identifies that we want to hit version 3 of the API.

The `params` are used to set the output languages. In this sample we're translating from English to Italian and German: `it` and `de`.

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Translate](#).

```
path = '/translate?api-version=3.0'
params = '&to=de&to=it'
constructed_url = endpoint + path + params
```

Add headers

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request. For more information, see [Authentication](#).

Copy this code snippet into your project:

```
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Create a request to translate text

Define the string (or strings) that you want to translate:

```
body = [{
    'text': 'Hello World!'
}]
```

Next, we'll create a POST request using the `requests` module. It takes three arguments: the concatenated URL, the request headers, and the request body:

```
request = requests.post(constructed_url, headers=headers, json=body)
response = request.json()
```

Print the response

The last step is to print the results. This code snippet prettifies the results by sorting the keys, setting indentation, and declaring item and key separators.

```
print(json.dumps(response, sort_keys=True, indent=4,
                  ensure_ascii=False, separators=(',', ' ')))
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
python translate-text.py
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "translations": [
      {
        "text": "Hallo Welt!",
        "to": "de"
      },
      {
        "text": "Salve, mondo!",
        "to": "it"
      }
    ]
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Node 8.12.x or later](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor, or a new folder with a file named `translate-text.js` on your desktop. Then copy this code snippet into your project/file:

```
const request = require('request');
const uuidv4 = require('uuid/v4');
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
npm install request uuidv4 .
```

These modules are required to construct the HTTP request, and create a unique identifier for the `'X-ClientTraceId'` header.

Set the subscription key and endpoint

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
var key_var = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY';
if (!process.env[key_var]) {
  throw new Error('Please set/export the following environment variable: ' + key_var);
}
var subscriptionKey = process.env[key_var];
var endpoint_var = 'TRANSLATOR_TEXT_ENDPOINT';
if (!process.env[endpoint_var]) {
  throw new Error('Please set/export the following environment variable: ' + endpoint_var);
}
var endpoint = process.env[endpoint_var];
```

Configure the request

The `request()` method, made available through the request module, allows us to pass the HTTP method, URL, request params, headers, and the JSON body as an `options` object. In this code snippet, we'll configure the request:

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Translate](#).

```
let options = {
  method: 'POST',
  baseUrl: endpoint,
  url: 'translate',
  qs: {
    'api-version': '3.0',
    'to': ['de', 'it']
  },
  headers: {
    'Ocp-Apim-Subscription-Key': subscriptionKey,
    'Content-type': 'application/json',
    'X-ClientTraceId': uuidv4().toString()
  },
  body: [{
    'text': 'Hello World!'
  }],
  json: true,
};
```

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request.

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request headers.

For more information, see [Authentication](#).

Make the request and print the response

Next, we'll create the request using the `request()` method. It takes the `options` object that we created in the previous section as the first argument, then prints the prettified JSON response.

```
request(options, function(err, res, body){
  console.log(JSON.stringify(body, null, 4));
});
```

NOTE

In this sample, we're defining the HTTP request in the `options` object. However, the request module also supports convenience methods, like `.post` and `.get`. For more information, see [convenience methods](#).

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response.

Now it's time to run your program:

```
node translate-text.js
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "translations": [
      {
        "text": "Hallo Welt!",
        "to": "de"
      },
      {
        "text": "Salve, mondo!",
        "to": "it"
      }
    ]
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Go](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.

- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Go project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `translate-text.go`.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
)
```

Create the main function

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
func main() {
    /*
     * Read your subscription key from an env variable.
     * Please note: You can replace this code block with
     * var subscriptionKey = "YOUR_SUBSCRIPTION_KEY" if you don't
     * want to use env variables. If so, be sure to delete the "os" import.
     */
    if "" == os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_SUBSCRIPTION_KEY.")
    }
    subscriptionKey := os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY")
    if "" == os.Getenv("TRANSLATOR_TEXT_ENDPOINT") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_ENDPOINT.")
    }
    endpoint := os.Getenv("TRANSLATOR_TEXT_ENDPOINT")
    uri := endpoint + "/translate?api-version=3.0"
    /*
     * This calls our breakSentence function, which we'll
     * create in the next section. It takes a single argument,
     * the subscription key.
     */
    translate(subscriptionKey, uri)
}
```

Create a function to translate text

Let's create a function to translate text. This function will take a single argument, your Translator Text subscription key.

```
func translate(subscriptionKey string, uri string) {
    /*
        * In the next few sections, we'll add code to this
        * function to make a request and handle the response.
        */
}
```

Next, let's construct the URL. The URL is built using the `Parse()` and `Query()` methods. You'll notice that parameters are added with the `Add()` method. In this sample, you're translating from English to German and Italian: `de` and `it`.

Copy this code into the `translate` function.

```
// Build the request URL. See: https://golang.org/pkg/net/url/#example_URL_Parse
u, _ := url.Parse(uri)
q := u.Query()
q.Add("to", "de")
q.Add("to", "it")
u.RawQuery = q.Encode()
```

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Translate](#).

Create a struct for your request body

Next, create an anonymous structure for the request body and encode it as JSON with `json.Marshal()`. Add this code to the `translate` function.

```
// Create an anonymous struct for your request body and encode it to JSON
body := []struct {
    Text string
}{
    {Text: "Hello, world!"},
}
b, _ := json.Marshal(body)
```

Build the request

Now that you've encoded the request body as JSON, you can build your POST request, and call the Translator Text API.

```
// Build the HTTP POST request
req, err := http.NewRequest("POST", u.String(), bytes.NewBuffer(b))
if err != nil {
    log.Fatal(err)
}
// Add required headers to the request
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)
req.Header.Add("Content-Type", "application/json")

// Call the Translator Text API
res, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Handle and print the response

Add this code to the `translate` function to decode the JSON response, and then format and print the result.

```
// Decode the JSON response
var result interface{}
if err := json.NewDecoder(res.Body).Decode(&result); err != nil {
    log.Fatal(err)
}
// Format and print the response to terminal
prettyJSON, _ := json.MarshalIndent(result, "", " ")
fmt.Printf("%s\n", prettyJSON)
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
go run translate-text.go
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

A successful response is returned in JSON as shown in the following example:


```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "translations": [
      {
        "text": "Hallo Welt!",
        "to": "de"
      },
      {
        "text": "Salve, mondo!",
        "to": "it"
      }
    ]
  }
]
```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

See also

- [Transliterate text](#)
- [Identify the language by input](#)
- [Get alternate translations](#)
- [Get a list of supported languages](#)
- [Determine sentence lengths from an input](#)

Quickstart: Use the Translator Text API to transliterate text

12/10/2019 • 19 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to transliterate (convert) text from one script to another using the Translator Text REST API. In the sample provided, Japanese is transliterated to use the Latin alphabet.

This quickstart requires an [Azure Cognitive Services account](#) with a Translator Text resource. If you don't have an account, you can use the [free trial](#) to get a subscription key.

Prerequisites

This quickstart requires:

- C# 7.1 or later
- [.NET SDK](#)
- [Json.NET NuGet Package](#)
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a .NET Core project

Open a new command prompt (or terminal session) and run these commands:

```
dotnet new console -o transliterate-sample
cd transliterate-sample
```

The first command does two things. It creates a new .NET console application, and creates a directory named `transliterate-sample`. The second command changes to the directory for your project.

Next, you'll need to install Json.Net. From your project's directory, run:

```
dotnet add package Newtonsoft.Json --version 11.0.2
```

Select the C# language version

This quickstart requires C# 7.1 or later. There are a few ways to change the C# version for your project. In this guide, we'll show you how to adjust the `transliterate-sample.csproj` file. For all available options, such as changing the language in Visual Studio, see [Select the C# language version](#).

Open your project, then open `transliterate-sample.csproj`. Make sure that `LangVersion` is set to 7.1 or later. If there isn't a property group for the language version, add these lines:

```
<PropertyGroup>
  <LangVersion>7.1</LangVersion>
</PropertyGroup>
```

Add required namespaces to your project

The `dotnet new console` command that you ran earlier created a project, including `Program.cs`. This file is where you'll put your application code. Open `Program.cs`, and replace the existing using statements. These statements ensure that you have access to all the types required to build and run the sample app.

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
// Install Newtonsoft.Json with NuGet
using Newtonsoft.Json;
```

Create classes for the JSON response

Next, we're going to create a class that's used when deserializing the JSON response returned by the Translator Text API.

```
/// <summary>
/// The C# classes that represents the JSON returned by the Translator Text API.
/// </summary>
public class TransliterationResult
{
    public string Text { get; set; }
    public string Script { get; set; }
}
```

Get subscription information from environment variables

Add the following lines to the `Program` class. These lines read your subscription key and endpoint from environment variables, and throws an error if you run into any issues.

```
private const string key_var = "TRANSLATOR_TEXT_SUBSCRIPTION_KEY";
private static readonly string subscriptionKey = Environment.GetEnvironmentVariable(key_var);

private const string endpoint_var = "TRANSLATOR_TEXT_ENDPOINT";
private static readonly string endpoint = Environment.GetEnvironmentVariable(endpoint_var);

static Program()
{
    if (null == subscriptionKey)
    {
        throw new Exception("Please set/export the environment variable: " + key_var);
    }
    if (null == endpoint)
    {
        throw new Exception("Please set/export the environment variable: " + endpoint_var);
    }
}
// The code in the next section goes here.
```

Create a function to transliterate text

Within the `Program` class, create an asynchronous function called `TransliterateTextRequest()`. This function takes four arguments: `subscriptionKey`, `endpoint`, `route`, and `inputText`.

```
static public async Task TransliterateTextRequest(string subscriptionKey, string endpoint, string route,
string inputText)
{
    /*
    * The code for your call to the translation service will be added to this
    * function in the next few sections.
    */
}
```

Serialize the translation request

Next, we need to create and serialize the JSON object that includes the text you want to translate. Keep in mind, you can pass more than one object in the `body`.

```
object[] body = new object[] { new { Text = inputText } };
var requestBody = JsonConvert.SerializeObject(body);
```

Instantiate the client and make a request

These lines instantiate the `HttpClient` and the `HttpRequestMessage`:

```
using (var client = new HttpClient())
using (var request = new HttpRequestMessage())
{
    // In the next few sections you'll add code to construct the request.
}
```

Construct the request and print the response

Inside the `HttpRequestMessage` you'll:

- Declare the HTTP method

- Construct the request URI
- Insert the request body (serialized JSON object)
- Add required headers
- Make an asynchronous request
- Print the response

Add this code to the `HttpRequestMessage`:

```
// Build the request.
// Set the method to Post.
request.Method = HttpMethod.Post;
// Construct the URI and add headers.
request.RequestUri = new Uri(endpoint + route);
request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");
request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

// Send the request and get response.
HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);
// Read response as a string.
string result = await response.Content.ReadAsStringAsync();
// Deserialize the response using the classes created earlier.
TransliterationResult[] deserializedOutput = JsonConvert.DeserializeObject<TransliterationResult[]>(result);
// Iterate over the deserialized results.
foreach (TransliterationResult o in deserializedOutput)
{
    Console.WriteLine("Transliterated to {0} script: {1}", o.Script, o.Text);
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Put it all together

The last step is to call `TransliterateTextRequest()` in the `Main` function. In this sample, we're transliterating from Japanese to latin script. Locate `static void Main(string[] args)` and replace it with this code:

```
static async Task Main(string[] args)
{
    // This is our main function.
    // Output languages are defined in the route.
    // For a complete list of options, see API reference.
    // https://docs.microsoft.com/azure/cognitive-services/translator/reference/v3-0-transliterate
    string route = "/transliterate?api-version=3.0&language=ja&fromScript=jpan&toScript=latn";
    string textToTransliterate = @"こんにちは";
    await TransliterateTextRequest(subscriptionKey, endpoint, route, textToTransliterate);
    Console.WriteLine("Press any key to continue.");
    Console.ReadKey();
}
```

You'll notice that in `Main`, you're declaring `subscriptionKey`, `endpoint`, `route`, and the script to transliterate `textToTransliterate`.

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to your project directory and run:

```
dotnet run
```

Sample response

After you run the sample, you should see the following printed to terminal:

```
Transliterated to latn script: Kon\'nichiwa
```

This message is built from the raw JSON, which will look like this:

```
[
  {
    "script": "latn",
    "text": "konnichiwa"
  }
]
```

Clean up resources

Make sure to remove any confidential information from your sample app's source code, like subscription keys.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [JDK 7 or later](#)
- [Gradle](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Initialize a project with Gradle

Let's start by creating a working directory for this project. From the command line (or terminal), run this

command:

```
mkdir transliterate-sample
cd transliterate-sample
```

Next, you're going to initialize a Gradle project. This command will create essential build files for Gradle, most importantly, the `build.gradle.kts`, which is used at runtime to create and configure your application. Run this command from your working directory:

```
gradle init --type basic
```

When prompted to choose a **DSL**, select **Kotlin**.

Configure the build file

Locate `build.gradle.kts` and open it with your favorite IDE or text editor. Then copy in this build configuration:

```
plugins {
    java
    application
}
application {
    mainClassName = "Transliterate"
}
repositories {
    mavenCentral()
}
dependencies {
    compile("com.squareup.okhttp:okhttp:2.5.0")
    compile("com.google.code.gson:gson:2.8.5")
}
```

Take note that this sample has dependencies on OkHttp for HTTP requests, and Gson to handle and parse JSON. If you'd like to learn more about build configurations, see [Creating New Gradle Builds](#).

Create a Java file

Let's create a folder for your sample app. From your working directory, run:

```
mkdir -p src\main\java
```

Next, in this folder, create a file named `Transliterate.java`.

Import required libraries

Open `Transliterate.java` and add these import statements:

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
import com.squareup.okhttp.*;
```

Define variables

First, you'll need to create a public class for your project:

```
public class Transliterate {  
    // All project code goes here...  
}
```

Add these lines to the `Transliterate` class. First, the subscription key and endpoint are being read from environment variables. Then, you'll notice that along with the `api-version`, two additional parameters have been appended to the `url`. These parameters are used to set the input language, and the scripts for transliteration. In this sample, it's set to Japanese (`jpan`) and Latin (`latn`).

```
private static String subscriptionKey = System.getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY");  
private static String endpoint = System.getenv("TRANSLATOR_TEXT_ENDPOINT");  
String url = endpoint + "/transliterate?api-version=3.0&language=ja&fromScript=jpan&toScript=latn";
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Create a client and build a request

Add this line to the `Transliterate` class to instantiate the `OkHttpClient`:

```
// Instantiates the OkHttpClient.  
OkHttpClient client = new OkHttpClient();
```

Next, let's build the POST request. Feel free to change the text for transliteration.

```
// This function performs a POST request.  
public String Post() throws IOException {  
    MediaType mediaType = MediaType.parse("application/json");  
    RequestBody body = RequestBody.create(mediaType,  
        "[{\n\t\t\"Text\": \"こんにちは\"\n}]");  
    Request request = new Request.Builder()  
        .url(url).post(body)  
        .addHeader("Ocp-Apim-Subscription-Key", subscriptionKey)  
        .addHeader("Content-type", "application/json").build();  
    Response response = client.newCall(request).execute();  
    return response.body().string();  
}
```

Create a function to parse the response

This simple function parses and prettifies the JSON response from the Translator Text service.

```
// This function prettifies the json response.  
public static String prettify(String json_text) {  
    JsonParser parser = new JsonParser();  
    JsonElement json = parser.parse(json_text);  
    Gson gson = new GsonBuilder().setPrettyPrinting().create();  
    return gson.toJson(json);  
}
```

Put it all together

The last step is to make a request and get a response. Add these lines to your project:

```
public static void main(String[] args) {
    try {
        Transliterate transliterateRequest = new Transliterate();
        String response = transliterateRequest.Post();
        System.out.println(prettify(response));
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to the root of your working directory and run:

```
gradle build
```

When the build completes, run:

```
gradle run
```

Sample response

```
[
  {
    "text": "konnichiwa",
    "script": "latn"
  }
]
```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Python 2.7.x or 3.x](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor, or a new folder with a file named `transliterate-text.py` on your desktop. Then copy this code snippet into your project/file:

```
# -*- coding: utf-8 -*-  
import os, requests, uuid, json
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
pip install requests uuid
```

The first comment tells your Python interpreter to use UTF-8 encoding. Then required modules are imported to read your subscription key from an environment variable, construct the http request, create a unique identifier, and handle the JSON response returned by the Translator Text API.

Set the subscription key, endpoint, and path

This sample will try to read your Translator Text subscription key and endpoint from the environment variables:

`TRANSLATOR_TEXT_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscription_key` and `endpoint` as a strings and comment out the conditional statements.

Copy this code into your project:

```
key_var_name = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY'  
if not key_var_name in os.environ:  
    raise Exception('Please set/export the environment variable: {}'.format(key_var_name))  
subscription_key = os.environ[key_var_name]  
  
endpoint_var_name = 'TRANSLATOR_TEXT_ENDPOINT'  
if not endpoint_var_name in os.environ:  
    raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))  
endpoint = os.environ[endpoint_var_name]
```

The Translator Text global endpoint is set as the `endpoint`. `path` sets the `transliterate` route and identifies that we want to hit version 3 of the API.

The `params` are used to set the input language, and the input and output scripts. In this sample, we're transliterating from Japanese to the Latin alphabet.

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Transliterate](#).

```
path = '/transliterate?api-version=3.0'
params = '&language=ja&fromScript=jpan&toScript=latn'
constructed_url = endpoint + path + params
```

Add headers

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request. For more information, see [Authentication](#).

Copy this code snippet into your project:

```
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Create a request to transliterate text

Define the string (or strings) that you want to transliterate:

```
# Transliterate "good afternoon" from source Japanese.
# Note: You can pass more than one object in body.
body = [{
    'text': 'こんにちは'
}]
```

Next, we'll create a POST request using the `requests` module. It takes three arguments: the concatenated URL, the request headers, and the request body:

```
request = requests.post(constructed_url, headers=headers, json=body)
response = request.json()
```

Print the response

The last step is to print the results. This code snippet prettifies the results by sorting the keys, setting indentation, and declaring item and key separators.

```
print(json.dumps(response, sort_keys=True, indent=4,
                  ensure_ascii=False, separators=(',', ' ')))
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
python transliterate-text.py
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "script": "latn",
    "text": "konnichiwa"
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Node 8.12.x or later](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor, or a new folder with a file named `translate-text.js` on your desktop. Then copy this code snippet into your project/file:

```
const request = require('request');
const uuidv4 = require('uuid/v4');
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
npm install request uuidv4 .
```

These modules are required to construct the HTTP request, and create a unique identifier for the 'X-ClientTraceId' header.

Set the subscription key and endpoint

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
var key_var = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY';
if (!process.env[key_var]) {
    throw new Error('Please set/export the following environment variable: ' + key_var);
}
var subscriptionKey = process.env[key_var];
var endpoint_var = 'TRANSLATOR_TEXT_ENDPOINT';
if (!process.env[endpoint_var]) {
    throw new Error('Please set/export the following environment variable: ' + endpoint_var);
}
var endpoint = process.env[endpoint_var];
```

Configure the request

The `request()` method, made available through the request module, allows us to pass the HTTP method, URL, request params, headers, and the JSON body as an `options` object. In this code snippet, we'll configure the request:

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Transliterate](#).

```
let options = {
  method: 'POST',
  baseUrl: endpoint,
  url: 'transliterate',
  qs: {
    'api-version': '3.0',
    'language': 'ja',
    'fromScript': 'jpan',
    'toScript': 'latn'
  },
  headers: {
    'Ocp-Apim-Subscription-Key': subscriptionKey,
    'Content-type': 'application/json',
    'X-ClientTraceId': uuidv4().toString()
  },
  body: [{
    'text': 'こんにちは'
  }],
  json: true,
};
```

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request.

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request headers.

For more information, see [Authentication](#).

Make the request and print the response

Next, we'll create the request using the `request()` method. It takes the `options` object that we created in the previous section as the first argument, then prints the prettified JSON response.

```
request(options, function(err, res, body){
  console.log(JSON.stringify(body, null, 4));
});
```

NOTE

In this sample, we're defining the HTTP request in the `options` object. However, the request module also supports convenience methods, like `.post` and `.get`. For more information, see [convenience methods](#).

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
node transliterate-text.js
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "script": "latn",
    "text": "konnichiwa"
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Go](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Go project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `transliterate-text.go`.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
)
```

Create the main function

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
func main() {
    /*
     * Read your subscription key from an env variable.
     * Please note: You can replace this code block with
     * var subscriptionKey = "YOUR_SUBSCRIPTION_KEY" if you don't
     * want to use env variables. If so, be sure to delete the "os" import.
     */
    if "" == os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_SUBSCRIPTION_KEY.")
    }
    subscriptionKey := os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY")
    if "" == os.Getenv("TRANSLATOR_TEXT_ENDPOINT") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_ENDPOINT.")
    }
    endpoint := os.Getenv("TRANSLATOR_TEXT_ENDPOINT")
    uri := endpoint + "/transliterate?api-version=3.0"
    /*
     * This calls our breakSentence function, which we'll
     * create in the next section. It takes a single argument,
     * the subscription key.
     */
    transliterate(subscriptionKey, uri)
}
```

Create a function to transliterate text

Let's create a function to transliterate text. This function will take a single argument, your Translator Text subscription key.

```
func transliterate(subscriptionKey string, uri string) {
    /*
     * In the next few sections, we'll add code to this
     * function to make a request and handle the response.
     */
}
```

Next, let's construct the URL. The URL is built using the `Parse()` and `Query()` methods. You'll notice that parameters are added with the `Add()` method. In this sample, we're transliterating from Japanese to the Latin alphabet.

Copy this code into the `transliterate` function.

```
// Build the request URL. See: https://golang.org/pkg/net/url/#example_URL_Parse
u, _ := url.Parse(uri)
q := u.Query()
q.Add("language", "ja")
q.Add("fromScript", "jpan")
q.Add("toScript", "latn")
u.RawQuery = q.Encode()
```

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Transliterate](#).

Create a struct for your request body

Next, create an anonymous structure for the request body and encode it as JSON with `json.Marshal()`. Add this code to the `transliterate` function.

```
// Create an anonymous struct for your request body and encode it to JSON
body := []struct {
    Text string
}{
    {Text: "こんにちは"},
}
b, _ := json.Marshal(body)
```

Build the request

Now that you've encoded the request body as JSON, you can build your POST request, and call the Translator Text API.

```
// Build the HTTP POST request
req, err := http.NewRequest("POST", u.String(), bytes.NewBuffer(b))
if err != nil {
    log.Fatal(err)
}
// Add required headers to the request
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)
req.Header.Add("Content-Type", "application/json")

// Call the Translator Text API
res, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Handle and print the response

Add this code to the `transliterate` function to decode the JSON response, and then format and print the result.

```
// Decode the JSON response
var result interface{}
if err := json.NewDecoder(res.Body).Decode(&result); err != nil {
    log.Fatal(err)
}
// Format and print the response to terminal
prettyJSON, _ := json.MarshalIndent(result, "", " ")
fmt.Printf("%s\n", prettyJSON)
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
go run transliterate-text.go
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "script": "latn",
    "text": "konnichiwa"
  }
]
```

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

See also

- [Translate text](#)
- [Identify the language by input](#)
- [Get alternate translations](#)
- [Get a list of supported languages](#)
- [Determine sentence lengths from an input](#)

Quickstart: Use the Translator Text API to detect text language

12/10/2019 • 20 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to detect the language of provided text with the Translator Text REST API.

This quickstart requires an [Azure Cognitive Services account](#) with a Translator Text resource. If you don't have an account, you can use the [free trial](#) to get a subscription key.

Prerequisites

This quickstart requires:

- C# 7.1 or later
- [.NET SDK](#)
- [Json.NET NuGet Package](#)
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a .NET Core project

Open a new command prompt (or terminal session) and run these commands:

```
dotnet new console -o detect-sample
cd detect-sample
```

The first command does two things. It creates a new .NET console application, and creates a directory named `detect-sample`. The second command changes to the directory for your project.

Next, you'll need to install Json.Net. From your project's directory, run:

```
dotnet add package Newtonsoft.Json --version 11.0.2
```

Select the C# language version

This quickstart requires C# 7.1 or later. There are a few ways to change the C# version for your project. In this guide, we'll show you how to adjust the `detect-sample.csproj` file. For all available options, such as changing the language in Visual Studio, see [Select the C# language version](#).

Open your project, then open `detect-sample.csproj`. Make sure that `LangVersion` is set to 7.1 or later. If there isn't a property group for the language version, add these lines:

```
<PropertyGroup>
  <LangVersion>7.1</LangVersion>
</PropertyGroup>
```

Add required namespaces to your project

The `dotnet new console` command that you ran earlier created a project, including `Program.cs`. This file is where you'll put your application code. Open `Program.cs`, and replace the existing using statements. These statements ensure that you have access to all the types required to build and run the sample app.

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
// Install Newtonsoft.Json with NuGet
using Newtonsoft.Json;
```

Create classes for the JSON response

Next, we're going to create a class that's used when deserializing the JSON response returned by the Translator Text API.

```
/// <summary>
/// The C# classes that represents the JSON returned by the Translator Text API.
/// </summary>
public class DetectResult
{
    public string Language { get; set; }
    public float Score { get; set; }
    public bool IsTranslationSupported { get; set; }
    public bool IsTransliterationSupported { get; set; }
    public AltTranslations[] Alternatives { get; set; }
}
public class AltTranslations
{
    public string Language { get; set; }
    public float Score { get; set; }
    public bool IsTranslationSupported { get; set; }
    public bool IsTransliterationSupported { get; set; }
}
```

Get subscription information from environment variables

Add the following lines to the `Program` class. These lines read your subscription key and endpoint from environment variables, and throws an error if you run into any issues.

```
private const string key_var = "TRANSLATOR_TEXT_SUBSCRIPTION_KEY";
private static readonly string subscriptionKey = Environment.GetEnvironmentVariable(key_var);

private const string endpoint_var = "TRANSLATOR_TEXT_ENDPOINT";
private static readonly string endpoint = Environment.GetEnvironmentVariable(endpoint_var);

static Program()
{
    if (null == subscriptionKey)
    {
        throw new Exception("Please set/export the environment variable: " + key_var);
    }
    if (null == endpoint)
    {
        throw new Exception("Please set/export the environment variable: " + endpoint_var);
    }
}
// The code in the next section goes here.
```

Create a function to detect the source text's language

In the `Program` class, create a function called `DetectTextRequest()`. This class encapsulates the code used to call the Detect resource and prints the result to console.

```
static public async Task DetectTextRequest(string subscriptionKey, string endpoint, string route, string
inputText)
{
    /*
    * The code for your call to the translation service will be added to this
    * function in the next few sections.
    */
}
```

Serialize the detect request

Next, we need to create and serialize the JSON object that includes the text that will undergo language detection.

```
System.Object[] body = new System.Object[] { new { Text = inputText } };
var requestBody = JsonConvert.SerializeObject(body);
```

Instantiate the client and make a request

These lines instantiate the `HttpClient` and the `HttpRequestMessage`:

```
using (var client = new HttpClient())
using (var request = new HttpRequestMessage())
{
    // In the next few sections you'll add code to construct the request.
}
```

Construct the request and print the response

Inside the `HttpRequestMessage` you'll:

- Declare the HTTP method
- Construct the request URI

- Insert the request body (serialized JSON object)
- Add required headers
- Make an asynchronous request
- Print the response

Add this code to the `HttpRequestMessage`:

```
// Build the request.
request.Method = HttpMethod.Post;
// Construct the URI and add headers.
request.RequestUri = new Uri(endpoint + route);
request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");
request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

// Send the request and get response.
HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);
// Read response as a string.
string result = await response.Content.ReadAsStringAsync();
// Deserialize the response using the classes created earlier.
DetectResult[] deserializedOutput = JsonConvert.DeserializeObject<DetectResult[]>(result);
// Iterate over the deserialized response.
foreach (DetectResult o in deserializedOutput)
{
    Console.WriteLine("The detected language is '{0}'. Confidence is: {1}.\nTranslation supported: {2}.\nTransliteration supported: {3}.\n",
        o.Language, o.Score, o.IsTranslationSupported, o.IsTransliterationSupported);
    // Create a counter
    int counter = 0;
    // Iterate over alternate translations.
    foreach (AltTranslations a in o.Alternatives)
    {
        counter++;
        Console.WriteLine("Alternative {0}", counter);
        Console.WriteLine("The detected language is '{0}'. Confidence is: {1}.\nTranslation supported: {2}.\nTransliteration supported: {3}.\n",
            a.Language, a.Score, a.IsTranslationSupported, a.IsTransliterationSupported);
    }
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Put it all together

The last step is to call `DetectTextRequest()` in the `Main` function. Locate `static void Main(string[] args)` and replace it with this code:

```
static async Task Main(string[] args)
{
    // This is our main function.
    // Output languages are defined in the route.
    // For a complete list of options, see API reference.
    string route = "/detect?api-version=3.0";
    string detectSentenceText = @"How are you doing today? The weather is pretty pleasant. Have you been to the movies lately?";
    await DetectTextRequest(subscriptionKey, endpoint, route, detectSentenceText);
    Console.WriteLine("Press any key to continue.");
    Console.ReadKey();
}
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to your project directory and run:

```
dotnet run
```

Sample response

After you run the sample, you should see the following printed to terminal:

NOTE

Find the country/region abbreviation in this [list of languages](#).

```
The detected language is 'en'. Confidence is: 1.
Translation supported: True.
Transliteration supported: False.
```

Alternative 1

```
The detected language is 'fil'. Confidence is: 0.82.
Translation supported: True.
Transliteration supported: False.
```

Alternative 2

```
The detected language is 'ro'. Confidence is: 1.
Translation supported: True.
Transliteration supported: False.
```

This message is built from the raw JSON, which will look like this:

```
[
  {
    "language": "en",
    "score": 1.0,
    "isTranslationSupported": true,
    "isTransliterationSupported": false,
    "alternatives": [
      {
        "language": "fil",
        "score": 0.82,
        "isTranslationSupported": true,
        "isTransliterationSupported": false
      },
      {
        "language": "ro",
        "score": 1.0,
        "isTranslationSupported": true,
        "isTransliterationSupported": false
      }
    ]
  }
]
```

Clean up resources

Make sure to remove any confidential information from your sample app's source code, like subscription keys.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [JDK 7 or later](#)
- [Gradle](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Initialize a project with Gradle

Let's start by creating a working directory for this project. From the command line (or terminal), run this command:

```
mkdir detect-sample
cd detect-sample
```

Next, you're going to initialize a Gradle project. This command will create essential build files for Gradle, most importantly, the `build.gradle.kts`, which is used at runtime to create and configure your application. Run this command from your working directory:

```
gradle init --type basic
```

When prompted to choose a **DSL**, select **Kotlin**.

Configure the build file

Locate `build.gradle.kts` and open it with your favorite IDE or text editor. Then copy in this build configuration:


```
plugins {
    java
    application
}
application {
    mainClassName = "Detect"
}
repositories {
    mavenCentral()
}
dependencies {
    compile("com.squareup.okhttp:okhttp:2.5.0")
    compile("com.google.code.gson:gson:2.8.5")
}
```

Take note that this sample has dependencies on OkHttp for HTTP requests, and Gson to handle and parse JSON. If you'd like to learn more about build configurations, see [Creating New Gradle Builds](#).

Create a Java file

Let's create a folder for your sample app. From your working directory, run:

```
mkdir -p src/main/java
```

Next, in this folder, create a file named `Detect.java`.

Import required libraries

Open `Detect.java` and add these import statements:

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
import com.squareup.okhttp.*;
```

Define variables

First, you'll need to create a public class for your project:

```
public class Detect {
    // All project code goes here...
}
```

Add these lines to the `Detect` class. You'll notice the subscription key and endpoint are being read from environment variables:

```
private static String subscriptionKey = System.getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY");
private static String endpoint = System.getenv("TRANSLATOR_TEXT_ENDPOINT");
String url = endpoint + "/detect?api-version=3.0";
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Create a client and build a request

Add this line to the `Detect` class to instantiate the `OkHttpClient` :

```
// Instantiates the OkHttpClient.
OkHttpClient client = new OkHttpClient();
```

Next, let's build the POST request. Feel free to change the text for language detection.

```
// This function performs a POST request.
public String Post() throws IOException {
    MediaType mediaType = MediaType.parse("application/json");
    RequestBody body = RequestBody.create(mediaType,
        "[{\n\t\"Text\": \"Salve mondo!\n}]\");
    Request request = new Request.Builder()
        .url(url).post(body)
        .addHeader("Ocp-Apim-Subscription-Key", subscriptionKey)
        .addHeader("Content-type", "application/json").build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

Create a function to parse the response

This simple function parses and prettifies the JSON response from the Translator Text service.

```
// This function prettifies the json response.
public static String prettify(String json_text) {
    JsonParser parser = new JsonParser();
    JsonElement json = parser.parse(json_text);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    return gson.toJson(json);
}
```

Put it all together

The last step is to make a request and get a response. Add these lines to your project:

```
public static void main(String[] args) {
    try {
        Detect detectRequest = new Detect();
        String response = detectRequest.Post();
        System.out.println(prrettify(response));
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to the root of your working directory and run:

```
gradle build
```

When the build completes, run:

```
gradle run
```

Sample response

After you run the sample, you should see the following printed to terminal:

NOTE

Find the country/region abbreviation in this [list of languages](#).

```
[
  {
    "language": "it",
    "score": 1.0,
    "isTranslationSupported": true,
    "isTransliterationSupported": false,
    "alternatives": [
      {
        "language": "pt",
        "score": 1.0,
        "isTranslationSupported": true,
        "isTransliterationSupported": false
      },
      {
        "language": "en",
        "score": 1.0,
        "isTranslationSupported": true,
        "isTransliterationSupported": false
      }
    ]
  }
]
```

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Python 2.7.x or 3.x](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Python project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `detect.py`.

```
# -*- coding: utf-8 -*-
import os, requests, uuid, json
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
pip install requests uuid
```

The first comment tells your Python interpreter to use UTF-8 encoding. Then required modules are imported to read your subscription key from an environment variable, construct the http request, create a unique identifier, and handle the JSON response returned by the Translator Text API.

Set the subscription key, endpoint, and path

This sample will try to read your Translator Text subscription key and endpoint from the environment variables:

`TRANSLATOR_TEXT_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscription_key` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
key_var_name = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY'
if not key_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(key_var_name))
subscription_key = os.environ[key_var_name]

endpoint_var_name = 'TRANSLATOR_TEXT_ENDPOINT'
if not endpoint_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))
endpoint = os.environ[endpoint_var_name]
```

The Translator Text global endpoint is set as the `endpoint`. `path` sets the `detect` route and identifies that we want to hit version 3 of the API.

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Detect](#).

```
path = '/detect?api-version=3.0'
constructed_url = endpoint + path
```

Add headers

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request. For more information, see [Authentication](#).

Copy this code snippet into your project:

```
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Create a request to detect text language

Define the string (or strings) that you want to detect the language for:

```
# You can pass more than one object in body.
body = [{
    'text': 'Salve, mondo!'
}]
```

Next, we'll create a POST request using the `requests` module. It takes three arguments: the concatenated URL, the request headers, and the request body:

```
request = requests.post(constructed_url, headers=headers, json=body)
response = request.json()
```

Print the response

The last step is to print the results. This code snippet prettifies the results by sorting the keys, setting indentation, and declaring item and key separators.

```
print(json.dumps(response, sort_keys=True, indent=4,
                  ensure_ascii=False, separators=(',', ' ': ' ')))
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
python detect.py
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

After you run the sample, you should see the following printed to terminal:

NOTE

Find the country/region abbreviation in this [list of languages](#).

```
[
  {
    "alternatives": [
      {
        "isTranslationSupported": true,
        "isTransliterationSupported": false,
        "language": "pt",
        "score": 1.0
      },
      {
        "isTranslationSupported": true,
        "isTransliterationSupported": false,
        "language": "en",
        "score": 1.0
      }
    ],
    "isTranslationSupported": true,
    "isTransliterationSupported": false,
    "language": "it",
    "score": 1.0
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Node 8.12.x or later](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.

- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `detect.js`.

```
const request = require('request');
const uuidv4 = require('uuid/v4');
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
npm install request uuidv4
```

These modules are required to construct the HTTP request, and create a unique identifier for the `'X-ClientTraceId'` header.

Set the subscription key and endpoint

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
var key_var = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY';
if (!process.env[key_var]) {
    throw new Error('Please set/export the following environment variable: ' + key_var);
}
var subscriptionKey = process.env[key_var];
var endpoint_var = 'TRANSLATOR_TEXT_ENDPOINT';
if (!process.env[endpoint_var]) {
    throw new Error('Please set/export the following environment variable: ' + endpoint_var);
}
var endpoint = process.env[endpoint_var];
```

Configure the request

The `request()` method, made available through the request module, allows us to pass the HTTP method, URL, request params, headers, and the JSON body as an `options` object. In this code snippet, we'll configure the request:

```
let options = {
  method: 'POST',
  baseUrl: endpoint,
  url: 'detect',
  qs: {
    'api-version': '3.0',
  },
  headers: {
    'Ocp-Apim-Subscription-Key': subscriptionKey,
    'Content-type': 'application/json',
    'X-ClientTraceId': uuidv4().toString()
  },
  body: [{
    'text': 'Salve, mondo!'
  }],
  json: true,
};
```

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request.

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request headers.

For more information, see [Authentication](#).

Make the request and print the response

Next, we'll create the request using the `request()` method. It takes the `options` object that we created in the previous section as the first argument, then prints the prettified JSON response.

```
request(options, function(err, res, body){
  console.log(JSON.stringify(body, null, 4));
});
```

NOTE

In this sample, we're defining the HTTP request in the `options` object. However, the request module also supports convenience methods, like `.post` and `.get`. For more information, see [convenience methods](#).

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
node detect.js
```

Sample response

After you run the sample, you should see the following printed to terminal:

NOTE

Find the country/region abbreviation in this [list of languages](#).

```
[
  {
    "alternatives": [
      {
        "isTranslationSupported": true,
        "isTransliterationSupported": false,
        "language": "pt",
        "score": 1.0
      },
      {
        "isTranslationSupported": true,
        "isTransliterationSupported": false,
        "language": "en",
        "score": 1.0
      }
    ],
    "isTranslationSupported": true,
    "isTransliterationSupported": false,
    "language": "it",
    "score": 1.0
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Go](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.

- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Go project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `detect-language.go`.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
)
```

Create the main function

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
func main() {
    /*
     * Read your subscription key from an env variable.
     * Please note: You can replace this code block with
     * var subscriptionKey = "YOUR_SUBSCRIPTION_KEY" if you don't
     * want to use env variables. If so, be sure to delete the "os" import.
     */
    if "" == os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_SUBSCRIPTION_KEY.")
    }
    subscriptionKey := os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY")
    if "" == os.Getenv("TRANSLATOR_TEXT_ENDPOINT") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_ENDPOINT.")
    }
    endpoint := os.Getenv("TRANSLATOR_TEXT_ENDPOINT")
    uri := endpoint + "/detect?api-version=3.0"
    /*
     * This calls our breakSentence function, which we'll
     * create in the next section. It takes a single argument,
     * the subscription key.
     */
    detect(subscriptionKey, uri)
}
```

Create a function to detect the text language

Let's create a function to detect the text language. This function will take a single argument, your Translator Text subscription key.

```
func detect(subscriptionKey string, uri string) {
    /*
        * In the next few sections, we'll add code to this
        * function to make a request and handle the response.
        */
}
```

Next, let's construct the URL. The URL is built using the `Parse()` and `Query()` methods.

Copy this code into the `detect` function.

```
// Build the request URL. See: https://golang.org/pkg/net/url/#example_URL_Parse
u, _ := url.Parse(uri)
q := u.Query()
u.RawQuery = q.Encode()
```

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Detect](#).

Create a struct for your request body

Next, create an anonymous structure for the request body and encode it as JSON with `json.Marshal()`. Add this code to the `detect` function.

```
// Create an anonymous struct for your request body and encode it to JSON
body := []struct {
    Text string
}{
    {Text: "Salve, Mondo!"},
}
b, _ := json.Marshal(body)
```

Build the request

Now that you've encoded the request body as JSON, you can build your POST request, and call the Translator Text API.

```
// Build the HTTP POST request
req, err := http.NewRequest("POST", u.String(), bytes.NewBuffer(b))
if err != nil {
    log.Fatal(err)
}
// Add required headers to the request
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)
req.Header.Add("Content-Type", "application/json")

// Call the Translator Text API
res, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Handle and print the response

Add this code to the `detect` function to decode the JSON response, and then format and print the result.

```
// Decode the JSON response
var result interface{}
if err := json.NewDecoder(res.Body).Decode(&result); err != nil {
    log.Fatal(err)
}
// Format and print the response to terminal
prettyJSON, _ := json.MarshalIndent(result, "", " ")
fmt.Printf("%s\n", prettyJSON)
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
go run detect-language.go
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

After you run the sample, you should see the following printed to terminal:

NOTE

Find the country/region abbreviation in this [list of languages](#).

```
[
  {
    "alternatives": [
      {
        "isTranslationSupported": true,
        "isTransliterationSupported": false,
        "language": "pt",
        "score": 1
      },
      {
        "isTranslationSupported": true,
        "isTransliterationSupported": false,
        "language": "en",
        "score": 1
      }
    ],
    "isTranslationSupported": true,
    "isTransliterationSupported": false,
    "language": "it",
    "score": 1
  }
]
```

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

See also

- [Translate text](#)
- [Transliterate text](#)
- [Get alternate translations](#)
- [Get a list of supported languages](#)
- [Determine sentence lengths from an input](#)

Quickstart: Look up words with bilingual dictionary

12/10/2019 • 19 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to get alternate translations for a term, and also usage examples of those alternate translations, using the Translator Text API.

This quickstart requires an [Azure Cognitive Services account](#) with a Translator Text resource. If you don't have an account, you can use the [free trial](#) to get a subscription key.

Prerequisites

This quickstart requires:

- C# 7.1 or later
- [.NET SDK](#)
- [Json.NET NuGet Package](#)
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a .NET Core project

Open a new command prompt (or terminal session) and run these commands:

```
dotnet new console -o alternate-sample
cd alternate-sample
```

The first command does two things. It creates a new .NET console application, and creates a directory named `alternate-sample`. The second command changes to the directory for your project.

Next, you'll need to install Json.Net. From your project's directory, run:

```
dotnet add package Newtonsoft.Json --version 11.0.2
```

Add required namespaces to your project

The `dotnet new console` command that you ran earlier created a project, including `Program.cs`. This file is where you'll put your application code. Open `Program.cs`, and replace the existing using statements. These statements ensure that you have access to all the types required to build and run the sample app.

```
using System;
using System.Net.Http;
using System.Text;
using Newtonsoft.Json;
```

Get subscription information from environment variables

Add the following lines to the `Program` class. These lines read your subscription key and endpoint from environment variables, and throws an error if you run into any issues.

```
private const string key_var = "TRANSLATOR_TEXT_SUBSCRIPTION_KEY";
private static readonly string subscriptionKey = Environment.GetEnvironmentVariable(key_var);

private const string endpoint_var = "TRANSLATOR_TEXT_ENDPOINT";
private static readonly string endpoint = Environment.GetEnvironmentVariable(endpoint_var);

static Program()
{
    if (null == subscriptionKey)
    {
        throw new Exception("Please set/export the environment variable: " + key_var);
    }
    if (null == endpoint)
    {
        throw new Exception("Please set/export the environment variable: " + endpoint_var);
    }
}
// The code in the next section goes here.
```

Create a function to get alternate translations

Within the `Program` class, create a function called `AltTranslation`. This class encapsulates the code used to call the Dictionary resource and prints the result to console.

```
static void AltTranslation()
{
    /*
     * The code for your call to the translation service will be added to this
     * function in the next few sections.
     */
}
```

Construct the URI

Add these lines to the `AltTranslation` function. You'll notice that along with the `api-version`, two additional parameters have been declared. These parameters are used to set the translation input and output. In this sample, these are English (`en`) and Spanish (`es`).

```
string route = "/dictionary/lookup?api-version=3.0";
static string params_ = "from=en&to=es";
static string uri = endpoint + path + params_;
```

Next, we need to create and serialize the JSON object that includes the text you want to translate. Keep in mind, you can pass more than one object in the `body` array.

```
System.Object[] body = new System.Object[] { new { Text = @"Elephants" } };  
var requestBody = JsonConvert.SerializeObject(body);
```

Instantiate the client and make a request

These lines instantiate the `HttpClient` and the `HttpRequestMessage` :

```
using (var client = new HttpClient())  
using (var request = new HttpRequestMessage())  
{  
    // In the next few sections you'll add code to construct the request.  
}
```

Construct the request and print the response

Inside the `HttpRequestMessage` you'll:

- Declare the HTTP method
- Construct the request URI
- Insert the request body (serialized JSON object)
- Add required headers
- Make an asynchronous request
- Print the response

Add this code to the `HttpRequestMessage` :

```
// Set the method to POST  
request.Method = HttpMethod.Post;  
  
// Construct the full URI  
request.RequestUri = new Uri(uri);  
  
// Add the serialized JSON object to your request  
request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");  
  
// Add the authorization header  
request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);  
  
// Send request, get response  
var response = client.SendAsync(request).Result;  
var jsonResponse = response.Content.ReadAsStringAsync().Result;  
  
// Print the response  
Console.WriteLine(PrettyPrint(jsonResponse));  
Console.WriteLine("Press any key to continue.");
```

Add `PrettyPrint` to add formatting to your JSON response:

```
static string PrettyPrint(string s)  
{  
    return JsonConvert.SerializeObject(JsonConvert.DeserializeObject(s), Formatting.Indented);  
}
```


If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Put it all together

The last step is to call `AltTranslation()` in the `Main` function. Locate `static void Main(string[] args)` and add these lines:

```
AltTranslation();
Console.WriteLine("Press any key to continue.");
Console.ReadKey();
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to your project directory and run:

```
dotnet run
```

Sample response

```
[
  {
    "displaySource": "elephants",
    "normalizedSource": "elephants",
    "translations": [
      {
        "backTranslations": [
          {
            "displayText": "elephants",
            "frequencyCount": 1207,
            "normalizedText": "elephants",
            "numExamples": 5
          }
        ],
        "confidence": 1.0,
        "displayTarget": "elefantas",
        "normalizedTarget": "elefantas",
        "posTag": "NOUN",
        "prefixWord": ""
      }
    ]
  }
]
```

Clean up resources

Make sure to remove any confidential information from your sample app's source code, like subscription keys.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [JDK 7 or later](#)
- [Gradle](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Initialize a project with Gradle

Let's start by creating a working directory for this project. From the command line (or terminal), run this command:

```
mkdir alt-translation-sample
cd alt-translation-sample
```

Next, you're going to initialize a Gradle project. This command will create essential build files for Gradle, most importantly, the `build.gradle.kts`, which is used at runtime to create and configure your application. Run this command from your working directory:

```
gradle init --type basic
```

When prompted to choose a **DSL**, select **Kotlin**.

Configure the build file

Locate `build.gradle.kts` and open it with your favorite IDE or text editor. Then copy in this build configuration:

```

plugins {
    java
    application
}
application {
    mainClassName = "AltTranslation"
}
repositories {
    mavenCentral()
}
dependencies {
    compile("com.squareup.okhttp:okhttp:2.5.0")
    compile("com.google.code.gson:gson:2.8.5")
}

```

Take note that this sample has dependencies on OkHttp for HTTP requests, and Gson to handle and parse JSON. If you'd like to learn more about build configurations, see [Creating New Gradle Builds](#).

Create a Java file

Let's create a folder for your sample app. From your working directory, run:

```
mkdir -p src\main\java
```

Next, in this folder, create a file named `AltTranslation.java`.

Import required libraries

Open `AltTranslation.java` and add these import statements:

```

import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
import com.squareup.okhttp.*;

```

Define variables

First, you'll need to create a public class for your project:

```

public class AltTranslation {
    // All project code goes here...
}

```

Add these lines to the `AltTranslation` class. First, the subscription key and endpoint are being read from environment variables. Then, you'll notice that along with the `api-version`, two additional parameters have been appended to the `url`. These parameters are used to set the translation input and output. In this sample, these are English (`en`) and Spanish (`es`).

```

private static String subscriptionKey = System.getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY");
private static String endpoint = System.getenv("TRANSLATOR_TEXT_ENDPOINT");
String url = endpoint + "/dictionary/lookup?api-version=3.0&from=en&to=es";

```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Create a client and build a request

Add this line to the `AltTranslation` class to instantiate the `OkHttpClient` :

```
// Instantiates the OkHttpClient.
OkHttpClient client = new OkHttpClient();
```

Next, let's build the POST request. Feel free to change the text for translation.

```
// This function performs a POST request.
public String Post() throws IOException {
    MediaType mediaType = MediaType.parse("application/json");
    RequestBody body = RequestBody.create(mediaType,
        "[{\n\t\t\"Text\": \"Pineapples\"\n}]");
    Request request = new Request.Builder()
        .url(url).post(body)
        .addHeader("Ocp-Apim-Subscription-Key", subscriptionKey)
        .addHeader("Content-type", "application/json").build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

Create a function to parse the response

This simple function parses and prettifies the JSON response from the Translator Text service.

```
// This function prettifies the json response.
public static String prettify(String json_text) {
    JsonParser parser = new JsonParser();
    JsonElement json = parser.parse(json_text);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    return gson.toJson(json);
}
```

Put it all together

The last step is to make a request and get a response. Add these lines to your project:

```
public static void main(String[] args) {
    try {
        AltTranslation altTranslationRequest = new AltTranslation();
        String response = altTranslationRequest.Post();
        System.out.println(prettify(response));
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to the root of your working directory and run:

```
gradle build
```

When the build completes, run:

```
gradle run
```

Sample response

```
[
  {
    "normalizedSource": "pineapples",
    "displaySource": "pineapples",
    "translations": [
      {
        "normalizedTarget": "piñas",
        "displayTarget": "piñas",
        "posTag": "NOUN",
        "confidence": 0.7016,
        "prefixWord": "",
        "backTranslations": [
          {
            "normalizedText": "pineapples",
            "displayText": "pineapples",
            "numExamples": 5,
            "frequencyCount": 158
          },
          {
            "normalizedText": "cones",
            "displayText": "cones",
            "numExamples": 5,
            "frequencyCount": 13
          },
          {
            "normalizedText": "piña",
            "displayText": "piña",
            "numExamples": 3,
            "frequencyCount": 5
          },
          {
            "normalizedText": "ganks",
            "displayText": "ganks",
            "numExamples": 2,
            "frequencyCount": 3
          }
        ]
      }
    ],
  },
  {
    "normalizedTarget": "ananás",
    "displayTarget": "ananás",
    "posTag": "NOUN",
    "confidence": 0.2984,
    "prefixWord": "",
    "backTranslations": [
      {
        "normalizedText": "pineapples",
        "displayText": "pineapples",
        "numExamples": 2,
        "frequencyCount": 16
      }
    ]
  }
]
}
```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Python 2.7.x or 3.x](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Python project using your favorite IDE or editor, or create a new folder on your desktop. Copy this code snippet into your project/folder into a file named `dictionary-lookup.py`.

```
# -*- coding: utf-8 -*-
import os, requests, uuid, json
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
pip install requests uuid
```

The first comment tells your Python interpreter to use UTF-8 encoding. Then required modules are imported to read your subscription key from an environment variable, construct the http request, create a unique identifier, and handle the JSON response returned by the Translator Text API.

Set the subscription key, endpoint, and path

This sample will try to read your Translator Text subscription key and endpoint from the environment variables:

`TRANSLATOR_TEXT_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscription_key` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
key_var_name = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY'
if not key_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(key_var_name))
subscription_key = os.environ[key_var_name]

endpoint_var_name = 'TRANSLATOR_TEXT_ENDPOINT'
if not endpoint_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))
endpoint = os.environ[endpoint_var_name]
```

The Translator Text global endpoint is set as the `endpoint`. `path` sets the `dictionary/lookup` route and identifies that we want to hit version 3 of the API.

The `params` are used to set the source and output languages. In this sample we're using English and Spanish: `en` and `es`.

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Dictionary Lookup](#).

```
path = '/dictionary/lookup?api-version=3.0'
params = '&from=en&to=es'
constructed_url = endpoint + path + params
```

Add headers

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request. For more information, see [Authentication](#).

Copy this code snippet into your project:

```
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Create a request to find alternate translations

Define the string (or strings) that you want to find translations for:

```
# You can pass more than one object in body.
body = [{
    'text': 'Elephants'
}]
```

Next, we'll create a POST request using the `requests` module. It takes three arguments: the concatenated URL, the request headers, and the request body:

```
request = requests.post(constructed_url, headers=headers, json=body)
response = request.json()
```

Print the response

The last step is to print the results. This code snippet prettifies the results by sorting the keys, setting indentation, and declaring item and key separators.


```
print(json.dumps(response, sort_keys=True, indent=4,
                 ensure_ascii=False, separators=(',', ' ')))
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
python alt-translations.py
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "displaySource": "elephants",
    "normalizedSource": "elephants",
    "translations": [
      {
        "backTranslations": [
          {
            "displayText": "elephants",
            "frequencyCount": 1207,
            "normalizedText": "elephants",
            "numExamples": 5
          }
        ],
        "confidence": 1.0,
        "displayTarget": "elefantes",
        "normalizedTarget": "elefantes",
        "posTag": "NOUN",
        "prefixWord": ""
      }
    ]
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Node 8.12.x or later](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor, or create a new folder on your desktop. Copy this code snippet into your project/folder into a file named `alt-translations.js`.

```
const request = require('request');
const uuidv4 = require('uuid/v4');
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
npm install request uuidv4
```

These modules are required to construct the HTTP request, and create a unique identifier for the `'X-ClientTraceId'` header.

Set the subscription key and endpoint

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
var key_var = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY';
if (!process.env[key_var]) {
  throw new Error('Please set/export the following environment variable: ' + key_var);
}
var subscriptionKey = process.env[key_var];
var endpoint_var = 'TRANSLATOR_TEXT_ENDPOINT';
if (!process.env[endpoint_var]) {
  throw new Error('Please set/export the following environment variable: ' + endpoint_var);
}
var endpoint = process.env[endpoint_var];
```

Configure the request

The `request()` method, made available through the request module, allows us to pass the HTTP method, URL, request params, headers, and the JSON body as an `options` object. In this code snippet, we'll configure the

request:

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Dictionary Lookup](#).

```
let options = {
  method: 'POST',
  baseUrl: endpoint,
  url: 'dictionary/lookup',
  qs: {
    'api-version': '3.0',
    'from': 'en',
    'to': 'es'
  },
  headers: {
    'Ocp-Apim-Subscription-Key': subscriptionKey,
    'Content-type': 'application/json',
    'X-ClientTraceId': uuidv4().toString()
  },
  body: [{
    'text': 'Elephants'
  }],
  json: true,
};
```

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request.

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request headers.

For more information, see [Authentication](#).

Make the request and print the response

Next, we'll create the request using the `request()` method. It takes the `options` object that we created in the previous section as the first argument, then prints the prettified JSON response.

```
request(options, function(err, res, body){
  console.log(JSON.stringify(body, null, 4));
});
```

NOTE

In this sample, we're defining the HTTP request in the `options` object. However, the request module also supports convenience methods, like `.post` and `.get`. For more information, see [convenience methods](#).

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
node alt-translations.js
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "displaySource": "elephants",
    "normalizedSource": "elephants",
    "translations": [
      {
        "backTranslations": [
          {
            "displayText": "elephants",
            "frequencyCount": 1207,
            "normalizedText": "elephants",
            "numExamples": 5
          }
        ],
        "confidence": 1.0,
        "displayTarget": "elefantes",
        "normalizedTarget": "elefantes",
        "posTag": "NOUN",
        "prefixWord": ""
      }
    ]
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Go](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.

- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Go project using your favorite IDE / editor or new folder on your desktop. Then copy this code snippet into your project/folder in a file named `dictionaryLookup.go`.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
)
```

Create the main function

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
func main() {
    /*
    * Read your subscription key from an env variable.
    * Please note: You can replace this code block with
    * var subscriptionKey = "YOUR_SUBSCRIPTION_KEY" if you don't
    * want to use env variables. If so, be sure to delete the "os" import.
    */
    if "" == os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_SUBSCRIPTION_KEY.")
    }
    subscriptionKey := os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY")
    if "" == os.Getenv("TRANSLATOR_TEXT_ENDPOINT") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_ENDPOINT.")
    }
    endpoint := os.Getenv("TRANSLATOR_TEXT_ENDPOINT")
    uri := endpoint + "/dictionary/lookup?api-version=3.0"
    /*
    * This calls our breakSentence function, which we'll
    * create in the next section. It takes a single argument,
    * the subscription key.
    */
    dictionaryLookup(subscriptionKey, uri)
}
```

Create a function to get alternate translations

Let's create a function to get alternate translations. This function will take a single argument, your Translator Text subscription key.

```
func dictionaryLookup(subscriptionKey string, uri string) {
    /*
     * In the next few sections, we'll add code to this
     * function to make a request and handle the response.
     */
}
```

Next, let's construct the URL. The URL is built using the `Parse()` and `Query()` methods. You'll notice that parameters are added with the `Add()` method. In this sample, we're translating from English to Spanish.

Copy this code into the `altTranslations` function.

```
// Build the request URL. See: https://golang.org/pkg/net/url/#example_URL_Parse
u, _ := url.Parse(uri)
q := u.Query()
q.Add("from", "en")
q.Add("to", "es")
u.RawQuery = q.Encode()
```

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Dictionary Lookup](#).

Create a struct for your request body

Next, create an anonymous structure for the request body and encode it as JSON with `json.Marshal()`. Add this code to the `altTranslations` function.

```
// Create an anonymous struct for your request body and encode it to JSON
body := []struct {
    Text string
}{
    {Text: "Pineapples"},
}
b, _ := json.Marshal(body)
```

Build the request

Now that you've encoded the request body as JSON, you can build your POST request, and call the Translator Text API.

```
// Build the HTTP POST request
req, err := http.NewRequest("POST", u.String(), bytes.NewBuffer(b))
if err != nil {
    log.Fatal(err)
}
// Add required headers to the request
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)
req.Header.Add("Content-Type", "application/json")

// Call the Translator Text API
res, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Handle and print the response

Add this code to the `altTranslations` function to decode the JSON response, and then format and print the result.

```
// Decode the JSON response
var result interface{}
if err := json.NewDecoder(res.Body).Decode(&result); err != nil {
    log.Fatal(err)
}
// Format and print the response to terminal
prettyJSON, _ := json.MarshalIndent(result, "", " ")
fmt.Printf("%s\n", prettyJSON)
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
go run dictionaryLookup.go
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "displaySource": "pineapples",
    "normalizedSource": "pineapples",
    "translations": [
      {
        "backTranslations": [
          {
            "displayText": "pineapples",
            "frequencyCount": 158,
            "normalizedText": "pineapples",
            "numExamples": 5
          },
          {
            "displayText": "cones",
            "frequencyCount": 13,
            "normalizedText": "cones",
            "numExamples": 5
          },
          {
            "displayText": "piña",
            "frequencyCount": 5,
            "normalizedText": "piña",
            "numExamples": 3
          },
          {
            "displayText": "ganks",
            "frequencyCount": 3,
            "normalizedText": "ganks",
            "numExamples": 2
          }
        ],
        "confidence": 0.7016,
        "displayTarget": "piñas",
        "normalizedTarget": "piñas",
        "posTag": "NOUN",
        "prefixWord": ""
      },
      {
        "backTranslations": [
          {
            "displayText": "pineapples",
            "frequencyCount": 16,
            "normalizedText": "pineapples",
            "numExamples": 2
          }
        ],
        "confidence": 0.2984,
        "displayTarget": "ananás",
        "normalizedTarget": "ananás",
        "posTag": "NOUN",
        "prefixWord": ""
      }
    ]
  }
]
```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

See also

- [Translate text](#)
- [Transliterate text](#)
- [Identify the language by input](#)
- [Get a list of supported languages](#)
- [Determine sentence lengths from an input](#)

Quickstart: Use the Translator Text API to get a list of supported languages

12/10/2019 • 18 minutes to read • [Edit Online](#)

In this quickstart, you get a list of languages supported for translation, transliteration, and dictionary lookup using the Translator Text API.

Prerequisites

This quickstart requires:

- C# 7.1 or later
- [.NET SDK](#)
- [Json.NET NuGet Package](#)
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a .NET Core project

Open a new command prompt (or terminal session) and run these commands:

```
dotnet new console -o languages-sample
cd languages-sample
```

The first command does two things. It creates a new .NET console application, and creates a directory named `languages-sample`. The second command changes to the directory for your project.

Next, you'll need to install Json.Net. From your project's directory, run:

```
dotnet add package Newtonsoft.Json --version 11.0.2
```

Add required namespaces to your project

The `dotnet new console` command that you ran earlier created a project, including `Program.cs`. This file is where you'll put your application code. Open `Program.cs`, and replace the existing using statements. These statements ensure that you have access to all the types required to build and run the sample app.

```
using System;
using System.Net.Http;
using System.Text;
using Newtonsoft.Json;
```

Get endpoint information from an environment variable

Add the following lines to the `Program` class. These lines read your subscription key and endpoint from environment variables, and throws an error if you run into any issues.

```
private const string endpoint_var = "TRANSLATOR_TEXT_ENDPOINT";
private static readonly string endpoint = Environment.GetEnvironmentVariable(endpoint_var);

static Program()
{
    if (null == endpoint)
    {
        throw new Exception("Please set/export the environment variable: " + endpoint_var);
    }
}
```

Create a function to get a list of languages

In the `Program` class, create a function called `GetLanguages`. This class encapsulates the code used to call the Languages resource and prints the result to console.

```
static void GetLanguages()
{
    /*
     * The code for your call to the translation service will be added to this
     * function in the next few sections.
     */
}
```

Set the route

Add these lines to the `GetLanguages` function.

```
string route = "/languages?api-version=3.0";
```

Instantiate the client and make a request

These lines instantiate the `HttpClient` and the `HttpRequestMessage`:

```
using (var client = new HttpClient())
using (var request = new HttpRequestMessage())
{
    // In the next few sections you'll add code to construct the request.
}
```

Construct the request and print the response

Inside the `HttpRequestMessage` you'll:

- Declare the HTTP method
- Construct the request URI
- Add required headers
- Make an asynchronous request
- Print the response

Add this code to the `HttpRequestMessage` :

```
// Set the method to GET
request.Method = HttpMethod.Get;
// Construct the full URI
request.RequestUri = new Uri(endpoint + route);
// Send request, get response
var response = client.SendAsync(request).Result;
var jsonResponse = response.Content.ReadAsStringAsync().Result;
// Pretty print the response
Console.WriteLine(PrettyPrint(jsonResponse));
Console.WriteLine("Press any key to continue.");
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

To print the response with "Pretty Print" (formatting for the response), add this function to your Program class:

```
static string PrettyPrint(string s)
{
    return JsonConvert.SerializeObject(JsonConvert.DeserializeObject(s), Formatting.Indented);
}
```

Put it all together

The last step is to call `GetLanguages()` in the `Main` function. Locate `static void Main(string[] args)` and add these lines:

```
GetLanguages();
Console.ReadLine();
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to your project directory and run:

```
dotnet run
```

Sample response

Find the country/region abbreviation in this [list of languages](#).

```

{
  "translation": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr"
    },
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "dir": "rtl"
    },
    ...
  },
  "transliteration": {
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "scripts": [
        {
          "code": "Arab",
          "name": "Arabic",
          "nativeName": "العربية",
          "dir": "rtl",
          "toScripts": [
            {
              "code": "Latn",
              "name": "Latin",
              "nativeName": "اللاتينية",
              "dir": "ltr"
            }
          ]
        }
      ],
    },
    {
      "code": "Latn",
      "name": "Latin",
      "nativeName": "اللاتينية",
      "dir": "ltr",
      "toScripts": [
        {
          "code": "Arab",
          "name": "Arabic",
          "nativeName": "العربية",
          "dir": "rtl"
        }
      ]
    }
  ],
  },
  ...
},
"dictionary": {
  "af": {
    "name": "Afrikaans",
    "nativeName": "Afrikaans",
    "dir": "ltr",
    "translations": [
      {
        "name": "English",
        "nativeName": "English",
        "dir": "ltr",
        "code": "en"
      }
    ]
  },
  "ar": {
    "name": "Arabic",
    "nativeName": "العربية",
    "dir": "rtl",

```

```
        "translations": [
            {
                "name": "English",
                "nativeName": "English",
                "dir": "ltr",
                "code": "en"
            }
        ],
        ...
    }
}
```

Clean up resources

Make sure to remove any confidential information from your sample app's source code, like subscription keys.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [JDK 7 or later](#)
- [Gradle](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Initialize a project with Gradle

Let's start by creating a working directory for this project. From the command line (or terminal), run this command:

```
mkdir get-languages-sample
cd get-languages-sample
```

Next, you're going to initialize a Gradle project. This command will create essential build files for Gradle, most importantly, the `build.gradle.kts`, which is used at runtime to create and configure your application. Run this

command from your working directory:

```
gradle init --type basic
```

When prompted to choose a **DSL**, select **Kotlin**.

Configure the build file

Locate `build.gradle.kts` and open it with your favorite IDE or text editor. Then copy in this build configuration:

```
plugins {
    java
    application
}
application {
    mainClassName = "GetLanguages"
}
repositories {
    mavenCentral()
}
dependencies {
    compile("com.squareup.okhttp:okhttp:2.5.0")
    compile("com.google.code.gson:gson:2.8.5")
}
```

Take note that this sample has dependencies on OkHttp for HTTP requests, and Gson to handle and parse JSON. If you'd like to learn more about build configurations, see [Creating New Gradle Builds](#).

Create a Java file

Let's create a folder for your sample app. From your working directory, run:

```
mkdir -p src/main/java
```

Next, in this folder, create a file named `GetLanguages.java`.

Import required libraries

Open `GetLanguages.java` and add these import statements:

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
import com.squareup.okhttp.*;
```

Define variables

First, you'll need to create a public class for your project:

```
public class GetLanguages {
    // All project code goes here...
}
```

Add these lines to the `GetLanguages` class. You'll notice the subscription key and endpoint are being read from

environment variables:

```
private static String subscriptionKey = System.getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY");
private static String endpoint = System.getenv("TRANSLATOR_TEXT_ENDPOINT");
String url = endpoint + "/languages?api-version=3.0";
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Create a client and build a request

Add this line to the `GetLanguages` class to instantiate the `OkHttpClient`:

```
// Instantiates the OkHttpClient.
OkHttpClient client = new OkHttpClient();
```

Next, let's build the `GET` request.

```
// This function performs a GET request.
public String Get() throws IOException {
    Request request = new Request.Builder()
        .url(url).get()
        .addHeader("Content-type", "application/json").build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

Create a function to parse the response

This simple function parses and prettifies the JSON response from the Translator Text service.

```
// This function prettifies the json response.
public static String prettify(String json_text) {
    JsonParser parser = new JsonParser();
    JsonElement json = parser.parse(json_text);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    return gson.toJson(json);
}
```

Put it all together

The last step is to make a request and get a response. Add these lines to your project:

```
public static void main(String[] args) {
    try {
        GetLanguages getLanguagesRequest = new GetLanguages();
        String response = getLanguagesRequest.Get();
        System.out.println(prettify(response));
    } catch (Exception e) {
        System.out.println(e);
    }
}
```


Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to the root of your working directory and run:

```
gradle build
```

When the build completes, run:

```
gradle run
```

Sample response

Find the country/region abbreviation in this [list of languages](#).

A successful response is returned in JSON as shown in the following example:

```
{
  "translation": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr"
    },
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "dir": "rtl"
    },
    ...
  },
  "transliteration": {
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "scripts": [
        {
          "code": "Arab",
          "name": "Arabic",
          "nativeName": "العربية",
          "dir": "rtl",
          "toScripts": [
            {
              "code": "Latn",
              "name": "Latin",
              "nativeName": "اللاتينية",
              "dir": "ltr"
            }
          ]
        }
      ]
    },
    {
      "code": "Latn",
      "name": "Latin",
      "nativeName": "اللاتينية",
      "dir": "ltr",
      "toScripts": [
        {
          "code": "Arab",
          "name": "Arabic",
          "nativeName": "العربية",
          "dir": "rtl"
        }
      ]
    }
  ]
}
```

```

    ]
  }
]
},
...
},
"dictionary": {
  "af": {
    "name": "Afrikaans",
    "nativeName": "Afrikaans",
    "dir": "ltr",
    "translations": [
      {
        "name": "English",
        "nativeName": "English",
        "dir": "ltr",
        "code": "en"
      }
    ]
  },
  "ar": {
    "name": "Arabic",
    "nativeName": "العربية",
    "dir": "rtl",
    "translations": [
      {
        "name": "English",
        "nativeName": "English",
        "dir": "ltr",
        "code": "en"
      }
    ]
  },
  ...
}
}

```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Python 2.7.x or 3.x](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.

- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Python project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `get-languages.py`.

```
# -*- coding: utf-8 -*-
import os, requests, uuid, json
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
pip install requests uuid
```

The first comment tells your Python interpreter to use UTF-8 encoding. Then required modules are imported to read your subscription key from an environment variable, construct the http request, create a unique identifier, and handle the JSON response returned by the Translator Text API.

Set the endpoint and path

This sample will try to read your Translator Text endpoint from an environment variable:

`TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `endpoint` as a string and comment out the conditional statement.

```
endpoint_var_name = 'TRANSLATOR_TEXT_ENDPOINT'
if not endpoint_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))
endpoint = os.environ[endpoint_var_name]
```

The Translator Text global endpoint is set as the `endpoint`. `path` sets the `languages` route and identifies that we want to hit version 3 of the API.

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Languages](#).

```
path = '/languages?api-version=3.0'
constructed_url = endpoint + path
```

Add headers

The request to get supported languages doesn't require authentication. Set the `Content-type` as `application/json` and add `x-ClientTraceId` to uniquely identify your request.

Copy this code snippet into your project:

```
headers = {
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Create a request to get a list of supported languages

Let's create a GET request using the `requests` module. It takes two arguments: the concatenated URL, and the request headers:

```
request = requests.get(constructed_url, headers=headers)
response = request.json()
```

Print the response

The last step is to print the results. This code snippet prettifies the results by sorting the keys, setting indentation, and declaring item and key separators.

```
print(json.dumps(response, sort_keys=True, indent=4,
                  ensure_ascii=False, separators=(',', ' ': )))
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
python get-languages.py
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

Find the country/region abbreviation in this [list of languages](#).

This sample has been truncated to show a snippet of the result:

```
{
  "translation": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr"
    },
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "dir": "rtl"
    },
    ...
  },
  "transliteration": {
```

```

"ar": {
  "name": "Arabic",
  "nativeName": "العربية",
  "scripts": [
    {
      "code": "Arab",
      "name": "Arabic",
      "nativeName": "العربية",
      "dir": "rtl",
      "toScripts": [
        {
          "code": "Latn",
          "name": "Latin",
          "nativeName": "اللاتينية",
          "dir": "ltr"
        }
      ]
    }
  ],
  {
    "code": "Latn",
    "name": "Latin",
    "nativeName": "اللاتينية",
    "dir": "ltr",
    "toScripts": [
      {
        "code": "Arab",
        "name": "Arabic",
        "nativeName": "العربية",
        "dir": "rtl"
      }
    ]
  }
],
...
},
"dictionary": {
  "af": {
    "name": "Afrikaans",
    "nativeName": "Afrikaans",
    "dir": "ltr",
    "translations": [
      {
        "name": "English",
        "nativeName": "English",
        "dir": "ltr",
        "code": "en"
      }
    ]
  },
  "ar": {
    "name": "Arabic",
    "nativeName": "العربية",
    "dir": "rtl",
    "translations": [
      {
        "name": "English",
        "nativeName": "English",
        "dir": "ltr",
        "code": "en"
      }
    ]
  },
  ...
}
}

```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Node 8.12.x or later](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `get-languages.js`.

```
const request = require('request');
const uuidv4 = require('uuid/v4');
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
npm install request uuidv4
```

These modules are required to construct the HTTP request, and create a unique identifier for the `'X-ClientTraceId'` header.

Set the endpoint

This sample will try to read your Translator Text endpoint from an environment variable:

`TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `endpoint` as a string and comment out the conditional statement.

lorum ipsum

Configure the request

The `request()` method, made available through the request module, allows us to pass the HTTP method, URL, request params, headers, and the JSON body as an `options` object. In this code snippet, we'll configure the request:

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Languages](#).

```
let options = {
  method: 'GET',
  baseUrl: endpoint,
  url: 'languages',
  qs: {
    'api-version': '3.0',
  },
  headers: {
    'Content-type': 'application/json',
    'X-ClientTraceId': uuidv4().toString()
  },
  json: true,
};
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Make the request and print the response

Next, we'll create the request using the `request()` method. It takes the `options` object that we created in the previous section as the first argument, then prints the prettified JSON response.

```
request(options, function(err, res, body){
  console.log(JSON.stringify(body, null, 4));
});
```

NOTE

In this sample, we're defining the HTTP request in the `options` object. However, the request module also supports convenience methods, like `.post` and `.get`. For more information, see [convenience methods](#).

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
node get-languages.js
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

Find the country/region abbreviation in this [list of languages](#).

This sample has been truncated to show a snippet of the result:

```
{
  "translation": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr"
    },
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "dir": "rtl"
    },
    ...
  },
  "transliteration": {
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "scripts": [
        {
          "code": "Arab",
          "name": "Arabic",
          "nativeName": "العربية",
          "dir": "rtl",
          "toScripts": [
            {
              "code": "Latn",
              "name": "Latin",
              "nativeName": "اللاتينية",
              "dir": "ltr"
            }
          ]
        },
        {
          "code": "Latn",
          "name": "Latin",
          "nativeName": "اللاتينية",
          "dir": "ltr",
          "toScripts": [
            {
              "code": "Arab",
              "name": "Arabic",
              "nativeName": "العربية",
              "dir": "rtl"
            }
          ]
        }
      ]
    },
    ...
  },
  "dictionary": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr",
      "translations": [
        {
          "name": "English",
          "nativeName": "English",
          "dir": "ltr",
          ...
        }
      ]
    }
  }
}
```



```

        "code": "en"
    }
}
],
},
"ar": {
    "name": "Arabic",
    "nativeName": "العربية",
    "dir": "rtl",
    "translations": [
        {
            "name": "English",
            "nativeName": "English",
            "dir": "ltr",
            "code": "en"
        }
    ]
},
...
}

```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Go](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Go project using your favorite IDE or editor or new folder on your desktop. Then copy this code snippet into your project/folder in a file named `get-languages.go`.

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
)
```

Create the main function

Let's create the main function for our application. You'll notice it's a single line of code. That's because we're creating a single function to get and print the list of supported languages for Translator Text.

This sample will try to read your Translator Text endpoint from an environment variable:

`TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `endpoint` as a string and comment out the conditional statement.

Copy this code into your project:

```
func main() {
    if "" == os.Getenv("TRANSLATOR_TEXT_ENDPOINT") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_ENDPOINT.")
    }
    endpoint := os.Getenv("TRANSLATOR_TEXT_ENDPOINT")
    uri := endpoint + "/languages?api-version=3.0"
    getLanguages(uri)
}
```

Create a function to get a list of supported languages

Let's create a function to get a list of supported languages.

```
func getLanguages(uri string) {
    /*
     * In the next few sections, we'll add code to this
     * function to make a request and handle the response.
     */
}
```

Next, let's construct the URL. The URL is built using the `Parse()` and `Query()` methods.

Copy this code into the `getLanguages` function.

```
// Build the request URL. See: https://golang.org/pkg/net/url/#example_URL_Parse
u, _ := url.Parse(uri)
q := u.Query()
u.RawQuery = q.Encode()
```

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Languages](#).

Build the request

Now that you've encoded the request body as JSON, you can build your POST request, and call the Translator Text API.

```
// Build the HTTP GET request
req, err := http.NewRequest("GET", u.String(), nil)
if err != nil {
    log.Fatal(err)
}
// Add required headers
req.Header.Add("Content-Type", "application/json")

// Call the Translator Text API
res, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Handle and print the response

Add this code to the `getLanguages` function to decode the JSON response, and then format and print the result.

```
// Decode the JSON response
var result interface{}
if err := json.NewDecoder(res.Body).Decode(&result); err != nil {
    log.Fatal(err)
}
// Format and print the response to terminal
prettyJSON, _ := json.MarshalIndent(result, "", " ")
fmt.Printf("%s\n", prettyJSON)
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
go run get-languages.go
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

Find the country/region abbreviation in this [list of languages](#).

A successful response is returned in JSON as shown in the following example:

```
{
  "dictionary": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr",
      "translations": [
```

```

        translations : [
            {
                "name": "English",
                "nativeName": "English",
                "dir": "ltr",
                "code": "en"
            }
        ]
    },
    "ar": {
        "name": "Arabic",
        "nativeName": "العربية",
        "dir": "rtl",
        "translations": [
            {
                "name": "English",
                "nativeName": "English",
                "dir": "ltr",
                "code": "en"
            }
        ]
    },
    ...
},
"translation": {
    "af": {
        "name": "Afrikaans",
        "nativeName": "Afrikaans",
        "dir": "ltr"
    },
    "ar": {
        "name": "Arabic",
        "nativeName": "العربية",
        "dir": "rtl"
    },
    ...
},
"transliteration": {
    "ar": {
        "name": "Arabic",
        "nativeName": "العربية",
        "scripts": [
            {
                "code": "Arab",
                "name": "Arabic",
                "nativeName": "العربية",
                "dir": "rtl",
                "toScripts": [
                    {
                        "code": "Latn",
                        "name": "Latin",
                        "nativeName": "اللاتينية",
                        "dir": "ltr"
                    }
                ]
            }
        ]
    },
    {
        "code": "Latn",
        "name": "Latin",
        "nativeName": "اللاتينية",
        "dir": "ltr",
        "toScripts": [
            {
                "code": "Arab",
                "name": "Arabic",
                "nativeName": "العربية",
                "dir": "rtl"
            }
        ]
    }
]

```

```
    }  
  ]  
},  
...  
}  
}
```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

See also

- [Translate text](#)
- [Transliterate text](#)
- [Identify the language by input](#)
- [Get alternate translations](#)
- [Determine sentence lengths from an input](#)

Quickstart: Use the Translator Text API to determine sentence length

12/10/2019 • 20 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to determine the length of sentences using the Translator Text API.

This quickstart requires an [Azure Cognitive Services account](#) with a Translator Text resource. If you don't have an account, you can use the [free trial](#) to get a subscription key.

Prerequisites

This quickstart requires:

- C# 7.1 or later
- [.NET SDK](#)
- [Json.NET NuGet Package](#)
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a .NET Core project

Open a new command prompt (or terminal session) and run these commands:

```
dotnet new console -o sentences-sample
cd sentences-sample
```

The first command does two things. It creates a new .NET console application, and creates a directory named `sentences-sample`. The second command changes to the directory for your project.

Next, you'll need to install Json.Net. From your project's directory, run:

```
dotnet add package Newtonsoft.Json --version 11.0.2
```

Select the C# language version

This quickstart requires C# 7.1 or later. There are a few ways to change the C# version for your project. In this guide, we'll show you how to adjust the `sentences-sample.csproj` file. For all available options, such as changing the language in Visual Studio, see [Select the C# language version](#).

Open your project, then open `sentences-sample.csproj`. Make sure that `LangVersion` is set to 7.1 or later. If there isn't a property group for the language version, add these lines:

```
<PropertyGroup>
  <LangVersion>7.1</LangVersion>
</PropertyGroup>
```

Add required namespaces to your project

The `dotnet new console` command that you ran earlier created a project, including `Program.cs`. This file is where you'll put your application code. Open `Program.cs`, and replace the existing using statements. These statements ensure that you have access to all the types required to build and run the sample app.

```
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
// Install Newtonsoft.Json with NuGet
using Newtonsoft.Json;
```

Create classes for the JSON response

Next, we're going to create a class that's used when deserializing the JSON response returned by the Translator Text API.

```
/// <summary>
/// The C# classes that represents the JSON returned by the Translator Text API.
/// </summary>
public class BreakSentenceResult
{
    public int[] SentLen { get; set; }
    public DetectedLanguage DetectedLanguage { get; set; }
}

public class DetectedLanguage
{
    public string Language { get; set; }
    public float Score { get; set; }
}
```

Get subscription information from environment variables

Add the following lines to the `Program` class. These lines read your subscription key and endpoint from environment variables, and throws an error if you run into any issues.

```
private const string key_var = "TRANSLATOR_TEXT_SUBSCRIPTION_KEY";
private static readonly string subscriptionKey = Environment.GetEnvironmentVariable(key_var);

private const string endpoint_var = "TRANSLATOR_TEXT_ENDPOINT";
private static readonly string endpoint = Environment.GetEnvironmentVariable(endpoint_var);

static Program()
{
    if (null == subscriptionKey)
    {
        throw new Exception("Please set/export the environment variable: " + key_var);
    }
    if (null == endpoint)
    {
        throw new Exception("Please set/export the environment variable: " + endpoint_var);
    }
}
// The code in the next section goes here.
```

Create a function to determine sentence length

In the `Program` class, create a new function called `BreakSentenceRequest()`. This function takes four arguments: `subscriptionKey`, `endpoint`, `route`, and `inputText`.

```
static public async Task BreakSentenceRequest(string subscriptionKey, string endpoint, string route, string
inputText)
{
    /*
     * The code for your call to the translation service will be added to this
     * function in the next few sections.
     */
}
```

Serialize the break sentence request

Next, you need to create and serialize the JSON object that includes the text. Keep in mind, you can pass more than one object in the `body` array.

```
object[] body = new object[] { new { Text = inputText } };
var requestBody = JsonConvert.SerializeObject(body);
```

Instantiate the client and make a request

These lines instantiate the `HttpClient` and the `HttpRequestMessage`:

```
using (var client = new HttpClient())
using (var request = new HttpRequestMessage())
{
    // In the next few sections you'll add code to construct the request.
}
```

Construct the request and print the response

Inside the `HttpRequestMessage` you'll:

- Declare the HTTP method

- Construct the request URI
- Insert the request body (serialized JSON object)
- Add required headers
- Make an asynchronous request
- Print the response

Add this code to the `HttpRequestMessage` :

```
// Build the request.
// Set the method to POST
request.Method = HttpMethod.Post;
// Construct the URI and add headers.
request.RequestUri = new Uri(endpoint + route);
request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");
request.Headers.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

// Send the request and get response.
HttpResponseMessage response = await client.SendAsync(request).ConfigureAwait(false);
// Read response as a string.
string result = await response.Content.ReadAsStringAsync();
// Deserialize the response using the classes created earlier.
BreakSentenceResult[] deserializedOutput = JsonConvert.DeserializeObject<BreakSentenceResult[]>(result);
foreach (BreakSentenceResult o in deserializedOutput)
{
    Console.WriteLine("The detected language is '{0}'. Confidence is: {1}.", o.DetectedLanguage.Language,
o.DetectedLanguage.Score);
    Console.WriteLine("The first sentence length is: {0}", o.SentLen[0]);
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Put it all together

The last step is to call `BreakSentenceRequest()` in the `Main` function. Locate `static void Main(string[] args)` and replace it with this code:

```
static async Task Main(string[] args)
{
    // This is our main function.
    // Output languages are defined in the route.
    // For a complete list of options, see API reference.
    string route = "/breaksentence?api-version=3.0";
    // Feel free to use any string.
    string breakSentenceText = @"How are you doing today? The weather is pretty pleasant. Have you been to
the movies lately?";
    await BreakSentenceRequest(subscriptionKey, endpoint, route, breakSentenceText);
    Console.WriteLine("Press any key to continue.");
    Console.ReadKey();
}
```

You'll notice that in `Main`, you're declaring `subscriptionKey`, `endpoint`, `route`, and the text to evaluate `breakSentenceText`.

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to your project directory and run:

```
dotnet run
```

Sample response

After you run the sample, you should see the following printed to terminal:

```
The detected language is \'en\'. Confidence is: 1.  
The first sentence length is: 25
```

This message is built from the raw JSON, which will look like this:

```
[  
  {  
    "detectedLanguage":  
    {  
      "language": "en",  
      "score": 1.0  
    },  
    "sentLen": [25, 32, 35]  
  }  
]
```

Clean up resources

Make sure to remove any confidential information from your sample app's source code, like subscription keys.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [JDK 7 or later](#)
- [Gradle](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Initialize a project with Gradle

Let's start by creating a working directory for this project. From the command line (or terminal), run this command:

```
mkdir length-sentence-sample
cd length-sentence-sample
```

Next, you're going to initialize a Gradle project. This command will create essential build files for Gradle, most importantly, the `build.gradle.kts`, which is used at runtime to create and configure your application. Run this command from your working directory:

```
gradle init --type basic
```

When prompted to choose a **DSL**, select **Kotlin**.

Configure the build file

Locate `build.gradle.kts` and open it with your favorite IDE or text editor. Then copy in this build configuration:

```
plugins {
    java
    application
}
application {
    mainClassName = "BreakSentence"
}
repositories {
    mavenCentral()
}
dependencies {
    compile("com.squareup.okhttp:okhttp:2.5.0")
    compile("com.google.code.gson:gson:2.8.5")
}
```

Take note that this sample has dependencies on OkHttp for HTTP requests, and Gson to handle and parse JSON. If you'd like to learn more about build configurations, see [Creating New Gradle Builds](#).

Create a Java file

Let's create a folder for your sample app. From your working directory, run:

```
mkdir -p src/main/java
```

Next, in this folder, create a file named `BreakSentence.java`.

Import required libraries

Open `BreakSentence.java` and add these import statements:

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
import com.squareup.okhttp.*;
```

Define variables

First, you'll need to create a public class for your project:

```
public class BreakSentence {
    // All project code goes here...
}
```

Add these lines to the `BreakSentence` class. First, the subscription key and endpoint are being read from environment variables. Then, you'll notice that along with the `api-version`, you can define the input language. In this sample it's English.

```
private static String subscriptionKey = System.getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY");
private static String endpoint = System.getenv("TRANSLATOR_TEXT_ENDPOINT");
String url = endpoint + "/breaksentence?api-version=3.0&language=en";
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription.](#)

Create a client and build a request

Add this line to the `BreakSentence` class to instantiate the `OkHttpClient`:

```
// Instantiates the OkHttpClient.
OkHttpClient client = new OkHttpClient();
```

Next, let's build the POST request. Feel free to change the text. The text must be escaped.

```
// This function performs a POST request.
public String Post() throws IOException {
    MediaType mediaType = MediaType.parse("application/json");
    RequestBody body = RequestBody.create(mediaType,
        "[{\n\t\"Text\": \"How are you? I am fine. What did you do today?\n}]]");
    Request request = new Request.Builder()
        .url(url).post(body)
        .addHeader("Ocp-Apim-Subscription-Key", subscriptionKey)
        .addHeader("Content-type", "application/json").build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}
```

Create a function to parse the response

This simple function parses and prettifies the JSON response from the Translator Text service.

```
// This function prettifies the json response.
public static String prettify(String json_text) {
    JsonParser parser = new JsonParser();
    JsonElement json = parser.parse(json_text);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    return gson.toJson(json);
}
```

Put it all together

The last step is to make a request and get a response. Add these lines to your project:

```
public static void main(String[] args) {
    try {
        BreakSentence breakSentenceRequest = new BreakSentence();
        String response = BreakSentenceRequest.Post();
        System.out.println(prettify(response));
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Run the sample app

That's it, you're ready to run your sample app. From the command line (or terminal session), navigate to the root of your working directory and run:

```
gradle build
```

When the build completes, run:

```
gradle run
```

Sample response

A successful response is returned in JSON as shown in the following example:

```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "sentLen": [
      13,
      11,
      22
    ]
  }
]
```

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

Prerequisites

This quickstart requires:

- [Python 2.7.x or 3.x](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Python project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `sentence-length.py`.

```
# -*- coding: utf-8 -*-  
import os, requests, uuid, json
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
pip install requests uuid
```

The first comment tells your Python interpreter to use UTF-8 encoding. Then required modules are imported to read your subscription key from an environment variable, construct the http request, create a unique identifier, and handle the JSON response returned by the Translator Text API.

Set the subscription key, endpoint, and path

This sample will try to read your Translator Text subscription key and endpoint from the environment variables:

`TRANSLATOR_TEXT_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscription_key` and `endpoint` as a strings and comment out the conditional statements.

Copy this code into your project:

```
key_var_name = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY'
if not key_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(key_var_name))
subscription_key = os.environ[key_var_name]

endpoint_var_name = 'TRANSLATOR_TEXT_ENDPOINT'
if not endpoint_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(endpoint_var_name))
endpoint = os.environ[endpoint_var_name]
```

The Translator Text global endpoint is set as the `endpoint`. `path` sets the `breaksentence` route and identifies that we want to hit version 3 of the API.

The `params` in this sample are used to set the language of the provided text. `params` aren't required for the `breaksentence` route. If left out of the request, the API will try to detect the language of the provided text, and provide this information along with a confidence score in the response.

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Languages](#).

```
path = '/breaksentence?api-version=3.0'
params = '&language=en'
constructed_url = endpoint + path + params
```

Add headers

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request. For more information, see [Authentication](#).

Copy this code snippet into your project:

```
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-type': 'application/json',
    'X-ClientTraceId': str(uuid.uuid4())
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Create a request to determine sentence length

Define the sentence (or sentences) that you want to determine the length of:

```
# You can pass more than one object in body.
body = [{
    'text': 'How are you? I am fine. What did you do today?'
}]
```

Next, we'll create a POST request using the `requests` module. It takes three arguments: the concatenated URL,

the request headers, and the request body:

```
request = requests.post(constructed_url, headers=headers, json=body)
response = request.json()
```

Print the response

The last step is to print the results. This code snippet prettifies the results by sorting the keys, setting indentation, and declaring item and key separators.

```
print(json.dumps(response, sort_keys=True, indent=4,
                  ensure_ascii=False, separators=(',', ' ')))
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
python sentence-length.py
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "sentLen": [
      13,
      11,
      22
    ]
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Node 8.12.x or later](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `sentence-length.js`.

```
const request = require('request');  
const uuidv4 = require('uuid/v4');
```

NOTE

If you haven't used these modules you'll need to install them before running your program. To install these packages, run:

```
npm install request uuidv4
```

These modules are required to construct the HTTP request, and create a unique identifier for the `'X-ClientTraceId'` header.

Set the subscription key and endpoint

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
var key_var = 'TRANSLATOR_TEXT_SUBSCRIPTION_KEY';  
if (!process.env[key_var]) {  
  throw new Error('Please set/export the following environment variable: ' + key_var);  
}  
var subscriptionKey = process.env[key_var];  
var endpoint_var = 'TRANSLATOR_TEXT_ENDPOINT';  
if (!process.env[endpoint_var]) {  
  throw new Error('Please set/export the following environment variable: ' + endpoint_var);  
}  
var endpoint = process.env[endpoint_var];
```

Configure the request

The `request()` method, made available through the request module, allows us to pass the HTTP method, URL, request params, headers, and the JSON body as an `options` object. In this code snippet, we'll configure the

request:

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: Break Sentence](#).

```
let options = {
  method: 'POST',
  baseUrl: endpoint,
  url: 'breaksentence',
  qs: {
    'api-version': '3.0',
  },
  headers: {
    'Ocp-Apim-Subscription-Key': subscriptionKey,
    'Content-type': 'application/json',
    'X-ClientTraceId': uuidv4().toString()
  },
  body: [{
    'text': 'How are you? I am fine. What did you do today?'
  }],
  json: true,
};
```

The easiest way to authenticate a request is to pass in your subscription key as an `Ocp-Apim-Subscription-Key` header, which is what we use in this sample. As an alternative, you can exchange your subscription key for an access token, and pass the access token along as an `Authorization` header to validate your request.

If you are using a Cognitive Services multi-service subscription, you must also include the `Ocp-Apim-Subscription-Region` in your request headers.

For more information, see [Authentication](#).

Make the request and print the response

Next, we'll create the request using the `request()` method. It takes the `options` object that we created in the previous section as the first argument, then prints the prettified JSON response.

```
request(options, function(err, res, body){
  console.log(JSON.stringify(body, null, 4));
});
```

NOTE

In this sample, we're defining the HTTP request in the `options` object. However, the request module also supports convenience methods, like `.post` and `.get`. For more information, see [convenience methods](#).

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
node sentence-length.js
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "sentLen": [
      13,
      11,
      22
    ]
  }
]
```

Clean up resources

If you've hardcoded your subscription key into your program, make sure to remove the subscription key when you're finished with this quickstart.

Next steps

Take a look at the [API reference](#) to understand everything you can do with the Translator Text API.

[API reference](#)

Prerequisites

This quickstart requires:

- [Go](#)
- An Azure subscription - [Create one for free](#)

Set up

Create a Translator Text resource

Azure Cognitive Services are represented by Azure resources that you subscribe to. Create a resource for Translator Text using the [Azure portal](#) or [Azure CLI](#) on your local machine. You can also:

- Get a [trial key](#) valid for 7 days for free. After signing up, it will be available on the Azure website.
- View an existing resource in the [Azure portal](#).

After you get a key from your trial subscription or resource, create two [environment variables](#):

- `TRANSLATOR_TEXT_SUBSCRIPTION_KEY` - The subscription key for your Translator Text resource.
- `TRANSLATOR_TEXT_ENDPOINT` - The global endpoint for Translator Text. Use `https://api.cognitive.microsofttranslator.com/`.

Create a project and import required modules

Create a new Go project using your favorite IDE or editor. Then copy this code snippet into your project in a file named `sentence-length.go`.

```
package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
)
```

Create the main function

This sample will try to read your Translator Text subscription key and endpoint from these environment variables:

`TRANSLATOR_TEXT_SUBSCRIPTION_KEY` and `TRANSLATOR_TEXT_ENDPOINT`. If you're not familiar with environment variables, you can set `subscriptionKey` and `endpoint` as strings and comment out the conditional statements.

Copy this code into your project:

```
func main() {
    /*
     * Read your subscription key from an env variable.
     * Please note: You can replace this code block with
     * var subscriptionKey = "YOUR_SUBSCRIPTION_KEY" if you don't
     * want to use env variables. If so, be sure to delete the "os" import.
     */
    if "" == os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_SUBSCRIPTION_KEY.")
    }
    subscriptionKey := os.Getenv("TRANSLATOR_TEXT_SUBSCRIPTION_KEY")
    if "" == os.Getenv("TRANSLATOR_TEXT_ENDPOINT") {
        log.Fatal("Please set/export the environment variable TRANSLATOR_TEXT_ENDPOINT.")
    }
    endpoint := os.Getenv("TRANSLATOR_TEXT_ENDPOINT")
    uri := endpoint + "/breaksentence?api-version=3.0"
    /*
     * This calls our breakSentence function, which we'll
     * create in the next section. It takes a single argument,
     * the subscription key.
     */
    breakSentence(subscriptionKey, uri)
}
```

Create a function to determine sentence length

Let's create a function to determine sentence length. This function will take a single argument, your Translator Text subscription key.

```
func breakSentence(subscriptionKey string, uri string)
/*
 * In the next few sections, we'll add code to this
 * function to make a request and handle the response.
 */
}
```

Next, let's construct the URL. The URL is built using the `Parse()` and `Query()` methods. You'll notice that parameters are added with the `Add()` method.

Copy this code into the `breakSentence` function.

```
// Build the request URL. See: https://golang.org/pkg/net/url/#example_URL_Parse
u, _ := url.Parse(uri)
q := u.Query()
q.Add("languages", "en")
u.RawQuery = q.Encode()
```

NOTE

For more information about endpoints, routes, and request parameters, see [Translator Text API 3.0: BreakSentence](#).

Create a struct for your request body

Next, create an anonymous structure for the request body and encode it as JSON with `json.Marshal()`. Add this code to the `breakSentence` function.

```
// Create an anonymous struct for your request body and encode it to JSON
body := []struct {
    Text string
}{
    {Text: "How are you? I am fine. What did you do today?"},
}
b, _ := json.Marshal(body)
```

Build the request

Now that you've encoded the request body as JSON, you can build your POST request, and call the Translator Text API.

```
// Build the HTTP POST request
req, err := http.NewRequest("POST", u.String(), bytes.NewBuffer(b))
if err != nil {
    log.Fatal(err)
}
// Add required headers to the request
req.Header.Add("Ocp-Apim-Subscription-Key", subscriptionKey)
req.Header.Add("Content-Type", "application/json")

// Call the Translator Text API
res, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatal(err)
}
```

If you are using a Cognitive Services multi-service subscription, you must also include the

`Ocp-Apim-Subscription-Region` in your request parameters. [Learn more about authenticating with the multi-service subscription](#).

Handle and print the response

Add this code to the `breakSentence` function to decode the JSON response, and then format and print the result.

```
// Decode the JSON response
var result interface{}
if err := json.NewDecoder(res.Body).Decode(&result); err != nil {
    log.Fatal(err)
}
// Format and print the response to terminal
prettyJSON, _ := json.MarshalIndent(result, "", " ")
fmt.Printf("%s\n", prettyJSON)
```

Put it all together

That's it, you've put together a simple program that will call the Translator Text API and return a JSON response. Now it's time to run your program:

```
go run sentence-length.go
```

If you'd like to compare your code against ours, the complete sample is available on [GitHub](#).

Sample response

```
[
  {
    "detectedLanguage": {
      "language": "en",
      "score": 1.0
    },
    "sentLen": [
      13,
      11,
      22
    ]
  }
]
```

Next steps

Take a look at the API reference to understand everything you can do with the Translator Text API.

[API reference](#)

See also

- [Translate text](#)
- [Transliterate text](#)
- [Identify the language by input](#)
- [Get alternate translations](#)
- [Get a list of supported languages](#)

Tutorial: Create a translation app with WPF

12/10/2019 • 17 minutes to read • [Edit Online](#)

In this tutorial, you'll build a [Windows Presentation Foundation \(WPF\)](#) app that uses Azure Cognitive Services for text translation, language detection, and spell checking with a single subscription key. Specifically, your app will call APIs from Translator Text and [Bing Spell Check](#).

What is WPF? It's a UI framework that creates desktop client apps. The WPF development platform supports a broad set of app development features, including an app model, resources, controls, graphics, layout, data binding, documents, and security. It's a subset of the .NET Framework, so if you have previously built apps with the .NET Framework using ASP.NET or Windows Forms, the programming experience should be familiar. WPF uses the Extensible app Markup Language (XAML) to provide a declarative model for app programming, which we'll review in the coming sections.

In this tutorial, you'll learn how to:

- Create a WPF project in Visual Studio
- Add assemblies and NuGet packages to your project
- Create your app's UI with XAML
- Use the Translator Text API to get languages, translate text, and detect the source language
- Use the Bing Spell Check API to validate your input and improve translation accuracy
- Run your WPF app

Cognitive Services used in this tutorial

This list includes the Cognitive Services used in this tutorial. Follow the link to browse the API reference for each feature.

SERVICE	FEATURE	DESCRIPTION
Translator Text	Get Languages	Retrieve a complete list of supported languages for text translation.
Translator Text	Translate	Translate text into more than 60 languages.
Translator Text	Detect	Detect the language of the input text. Includes confidence score for detection.
Bing Spell Check	Spell Check	Correct spelling errors to improve translation accuracy.

Prerequisites

Before we continue, you'll need the following:

- An Azure Cognitive Services subscription. [Get a Cognitive Services key](#).
- A Windows machine
- [Visual Studio 2019](#) - Community or Enterprise

NOTE

We recommend creating the subscription in the West US region for this tutorial. Otherwise, you'll need to change endpoints and regions in the code as you work through this exercise.

Create a WPF app in Visual Studio

The first thing we need to do is set up our project in Visual Studio.

1. Open Visual Studio. Select **Create a new project**.
2. In **Create a new project**, locate and select **WPF App (.NET Framework)**. You can select C# from **Language** to narrow the options.
3. Select **Next**, and then name your project `MSTranslatorTextDemo`.
4. Set the framework version to **.NET Framework 4.7.2** or later, and select **Create**.

Configure your new project

WPF App (.NET Framework) C# Windows Desktop

Project name

MSTranslatorTextDemo

Location

C:\Users\translatorprojects\Source\Repos

Solution name ⓘ

MSTranslatorTextDemo

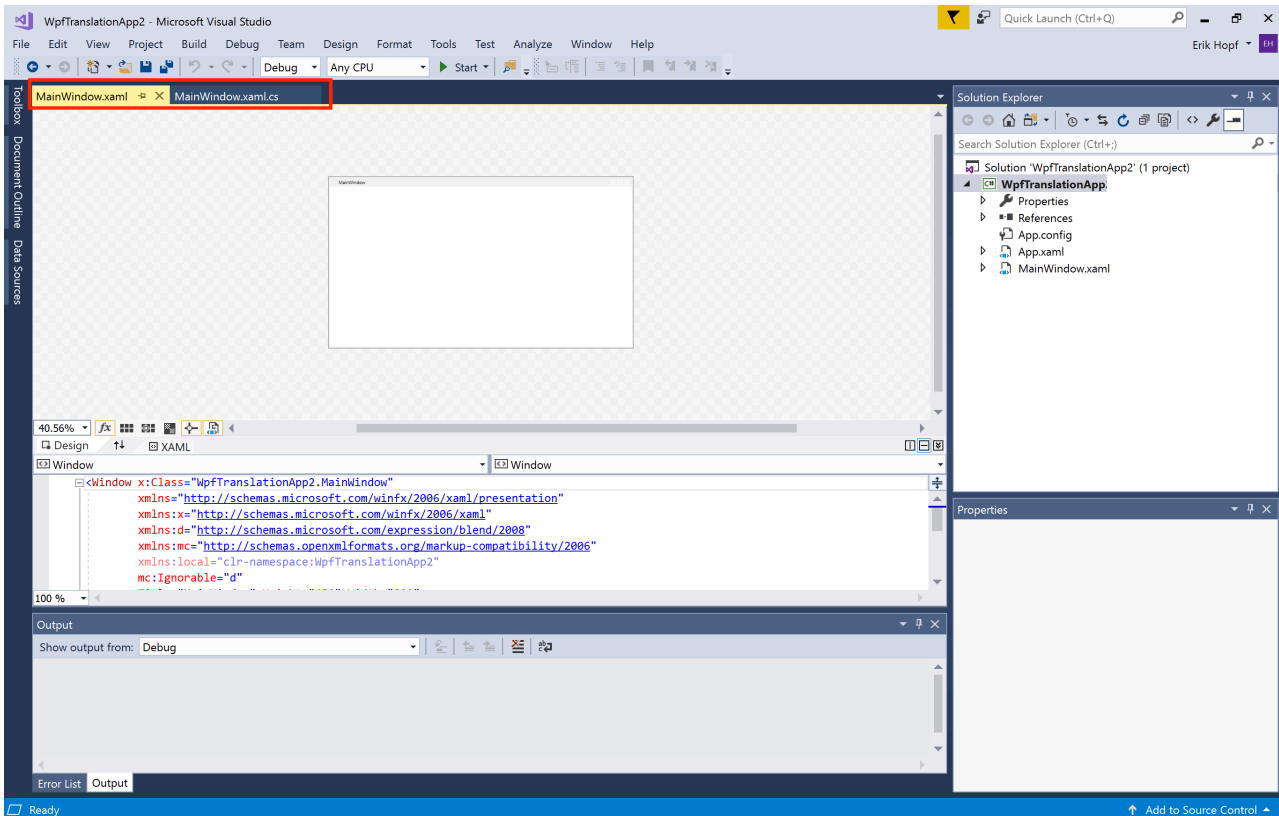
☐ Place solution and project in the same directory

Framework

.NET Framework 4.7.2

Back Create

Your project has been created. You'll notice that there are two tabs open: `MainWindow.xaml` and `MainWindow.xaml.cs`. Throughout this tutorial, we'll be adding code to these two files. We'll modify `MainWindow.xaml` for the app's user interface. We'll modify `MainWindow.xaml.cs` for our calls to Translator Text and Bing Spell Check.



In the next section, we're going to add assemblies and a NuGet package to our project for additional functionality, like JSON parsing.

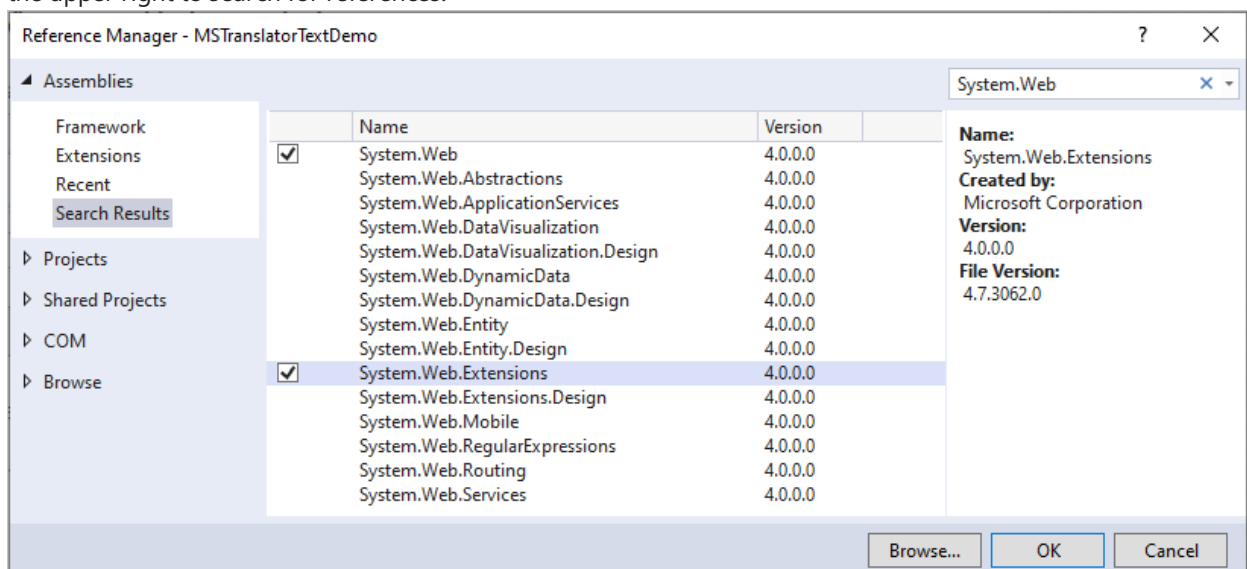
Add references and NuGet packages to your project

Our project requires a handful of .NET Framework assemblies and Newtonsoft.Json, which we'll install using the NuGet package manager.

Add .NET Framework assemblies

Let's add assemblies to our project to serialize and deserialize objects, and to manage HTTP requests and responses.

1. Locate your project in Visual Studio's Solution Explorer. Right-click your project, then select **Add > Reference**, which opens **Reference Manager**.
2. The **Assemblies** tab lists all .NET Framework assemblies that are available to reference. Use the search bar in the upper right to search for references.



3. Select the following references for your project:

- [System.Runtime.Serialization](#)
- [System.Web](#)
- System.Web.Extensions
- [System.Windows](#)

4. After you've added these references to your project, you can click **OK** to close **Reference Manager**.

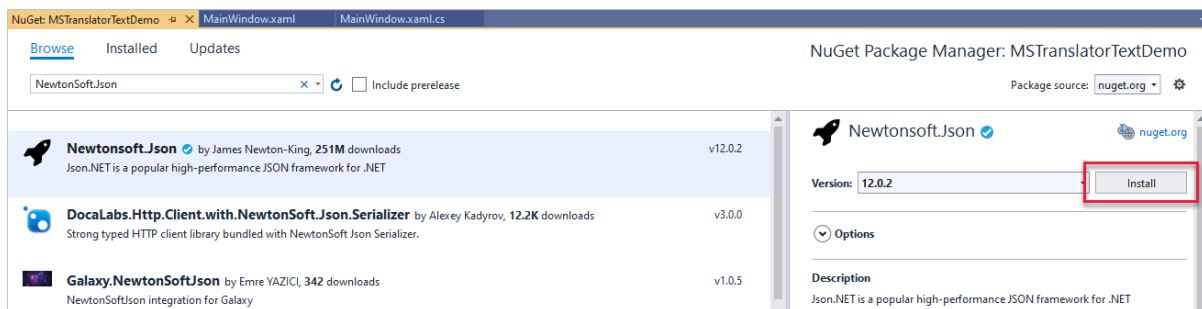
NOTE

If you'd like to learn more about assembly references, see [How to: Add or remove reference using the Reference Manager](#).

Install Newtonsoft.Json

Our app will use Newtonsoft.Json to deserialize JSON objects. Follow these instructions to install the package.

1. Locate your project in Visual Studio's Solution Explorer and right-click on your project. Select **Manage NuGet Packages**.
2. Locate and select the **Browse** tab.
3. Enter [NewtonSoft.Json](#) into the search bar.



4. Select the package and click **Install**.
5. When the installation is complete, close the tab.

Create a WPF form using XAML

To use your app, you're going to need a user interface. Using XAML, we'll create a form that allows users to select input and translation languages, enter text to translate, and displays the translation output.

Let's take a look at what we're building.

The user interface includes these components:

NAME	TYPE	DESCRIPTION
<code>FromLanguageComboBox</code>	ComboBox	Displays a list of the languages supported by Microsoft Translator for text translation. The user selects the language they are translating from.
<code>ToLanguageComboBox</code>	ComboBox	Displays the same list of languages as <code>FromComboBox</code> , but is used to select the language the user is translating to.
<code>TextToTranslate</code>	TextBox	Allows the user to enter text to be translated.
<code>TranslateButton</code>	Button	Use this button to translate text.
<code>TranslatedTextLabel</code>	Label	Displays the translation.
<code>DetectedLanguageLabel</code>	Label	Displays the detected language of the text to be translated (<code>TextToTranslate</code>).

NOTE

We're creating this form using the XAML source code, however, you can create the form with the editor in Visual Studio.

Let's add the code to our project.

1. In Visual Studio, select the tab for `MainWindow.xaml`.
2. Copy this code into your project, and then select **File > Save MainWindow.xaml** to save your changes.

```

<Window x:Class="MSTranslatorTextDemo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MSTranslatorTextDemo"
        mc:Ignorable="d"
        Title="Microsoft Translator" Height="400" Width="700" BorderThickness="0">
    <Grid>
        <Label x:Name="label" Content="Microsoft Translator" HorizontalAlignment="Left"
Margin="39,6,0,0" VerticalAlignment="Top" Height="49" FontSize="26.667"/>
        <TextBox x:Name="TextToTranslate" HorizontalAlignment="Left" Height="23" Margin="42,160,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="600" FontSize="14" TabIndex="3"/>
        <Label x:Name="EnterTextLabel" Content="Text to translate:" HorizontalAlignment="Left"
Margin="40,129,0,0" VerticalAlignment="Top" FontSize="14"/>
        <Label x:Name="toLabel" Content="Translate to:" HorizontalAlignment="Left" Margin="304,58,0,0"
VerticalAlignment="Top" FontSize="14"/>

        <Button x:Name="TranslateButton" Content="Translate" HorizontalAlignment="Left"
Margin="39,206,0,0" VerticalAlignment="Top" Width="114" Height="31" Click="TranslateButton_Click"
FontSize="14" TabIndex="4" IsDefault="True"/>
        <ComboBox x:Name="ToLanguageComboBox"
            HorizontalAlignment="Left"
            Margin="306,88,0,0"
            VerticalAlignment="Top"
            Width="175" FontSize="14" TabIndex="2">

    </ComboBox>
        <Label x:Name="fromLabel" Content="Translate from:" HorizontalAlignment="Left"
Margin="40,58,0,0" VerticalAlignment="Top" FontSize="14"/>
        <ComboBox x:Name="FromLanguageComboBox"
            HorizontalAlignment="Left"
            Margin="42,88,0,0"
            VerticalAlignment="Top"
            Width="175" FontSize="14" TabIndex="1"/>
        <Label x:Name="TranslatedTextLabel" Content="Translation is displayed here."
HorizontalAlignment="Left" Margin="39,255,0,0" VerticalAlignment="Top" Width="620" FontSize="14"
Height="85" BorderThickness="0"/>
        <Label x:Name="DetectedLanguageLabel" Content="Autodetected language is displayed here."
HorizontalAlignment="Left" Margin="39,288,0,0" VerticalAlignment="Top" Width="620" FontSize="14"
Height="84" BorderThickness="0"/>
    </Grid>
</Window>

```

You should now see a preview of the app's user interface in Visual Studio. It should look similar to the image above.

That's it, your form is ready. Now let's write some code to use Text Translation and Bing Spell Check.

NOTE

Feel free to tweak this form or create your own.

Create your app

`MainWindow.xaml.cs` contains the code that controls our app. In the next few sections, we're going to add code to populate our drop-down menus, and to call a handful of API exposed by Translator Text and Bing Spell Check.

- When the program starts and `MainWindow` is instantiated, the `Languages` method of the Translator Text API is called to retrieve and populate our language selection drop-downs. This happens once at the beginning of each session.
- When the **Translate** button is clicked, the user's language selection and text are retrieved, spell check is

performed on the input, and the translation and detected language are displayed for the user.

- The `Translate` method of the Translator Text API is called to translate text from `TextToTranslate`. This call also includes the `to` and `from` languages selected using the drop-down menus.
- The `Detect` method of the Translator Text API is called to determine the text language of `TextToTranslate`.
- Bing Spell Check is used to validate `TextToTranslate` and adjust misspellings.

All of our project is encapsulated in the `MainWindow : Window` class. Let's start by adding code to set your subscription key, declare endpoints for Translator Text and Bing Spell Check, and initialize the app.

1. In Visual Studio, select the tab for `MainWindow.xaml.cs`.
2. Replace the pre-populated `using` statements with the following.

```
using System;  
using System.Windows;  
using System.Net;  
using System.Net.Http;  
using System.IO;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using Newtonsoft.Json;
```

3. Locate the `MainWindow : Window` class, and replace it with this code:

```

{
    // This sample uses the Cognitive Services subscription key for all services. To learn more about
    // authentication options, see: https://docs.microsoft.com/azure/cognitive-services/authentication.
    const string COGNITIVE_SERVICES_KEY = "YOUR_COG_SERVICES_KEY";
    // Endpoints for Translator Text and Bing Spell Check
    public static readonly string TEXT_TRANSLATION_API_ENDPOINT =
"https://api.cognitive.microsofttranslator.com/{0}?api-version=3.0";
    const string BING_SPELL_CHECK_API_ENDPOINT =
"https://westus.api.cognitive.microsoft.com/bing/v7.0/spellcheck/";
    // An array of language codes
    private string[] languageCodes;

    // Dictionary to map language codes from friendly name (sorted case-insensitively on language name)
    private SortedDictionary<string, string> languageCodesAndTitles =
        new SortedDictionary<string, string>(Comparer<string>.Create((a, b) => string.Compare(a, b,
true)));

    // Global exception handler to display error message and exit
    private static void HandleExceptions(object sender, UnhandledExceptionEventArgs args)
    {
        Exception e = (Exception)args.ExceptionObject;
        MessageBox.Show("Caught " + e.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        System.Windows.Application.Current.Shutdown();
    }
    // MainWindow constructor
    public MainWindow()
    {
        // Display a message if unexpected error is encountered
        AppDomain.CurrentDomain.UnhandledException += new
        UnhandledExceptionEventHandler(HandleExceptions);

        if (COGNITIVE_SERVICES_KEY.Length != 32)
        {
            MessageBox.Show("One or more invalid API subscription keys.\n\n" +
                "Put your keys in the *_API_SUBSCRIPTION_KEY variables in MainWindow.xaml.cs.",
                "Invalid Subscription Key(s)", MessageBoxButton.OK, MessageBoxImage.Error);
            System.Windows.Application.Current.Shutdown();
        }
        else
        {
            // Start GUI
            InitializeComponent();
            // Get languages for drop-downs
            GetLanguagesForTranslate();
            // Populate drop-downs with values from GetLanguagesForTranslate
            PopulateLanguageMenus();
        }
    }
}
// NOTE:
// In the following sections, we'll add code below this.
}

```

4. Add your Cognitive Services subscription key and save.

In this code block, we've declared two member variables that contain information about available languages for translation:

VARIABLE	TYPE	DESCRIPTION
<code>languageCodes</code>	Array of strings	Caches the language codes. The Translator service uses short codes, such as <code>en</code> for English, to identify languages.

VARIABLE	TYPE	DESCRIPTION
<code>languageCodesAndTitles</code>	Sorted dictionary	Maps the "friendly" names in the user interface back to the short codes used in the API. Kept sorted alphabetically without regard for case.

Then, within the `MainWindow` constructor, we've added error handling with `HandleExceptions`. This error handling ensures that an alert is provided if an exception isn't handled. Then a check is run to confirm the subscription key provided is 32 characters in length. An error is thrown if the key is less than/greater than 32 characters.

If there are keys that are at least the right length, the `InitializeComponent()` call gets the user interface rolling by locating, loading, and instantiating the XAML description of the main app window.

Last, we've added code to call methods to retrieve languages for translation and to populate the drop-down menus for our app's user interface. Don't worry, we'll get to the code behind these calls soon.

Get supported languages

The Translator Text API currently supports more than 60 languages. Since new language support will be added over time, we recommend calling the Languages resource exposed by Translator Text rather than hardcoding the language list in your app.

In this section, we'll create a `GET` request to the Languages resource, specifying that we want a list of languages available for translation.

NOTE

The Languages resource allows you to filter language support with the following query parameters: transliteration, dictionary, and translation. For more information, see [API reference](#).

Before we go any further, let's take a look at a sample output for a call to the Languages resource:

```
{
  "translation": {
    "af": {
      "name": "Afrikaans",
      "nativeName": "Afrikaans",
      "dir": "ltr"
    },
    "ar": {
      "name": "Arabic",
      "nativeName": "العربية",
      "dir": "rtl"
    }
  }
  // Additional languages are provided in the full JSON output.
}
```

From this output, we can extract the language code and the `name` of a specific language. Our app uses `NewtonSoft.Json` to deserialize the JSON object (`JsonConvert.DeserializeObject`).

Picking up where we left off in the last section, let's add a method to get supported languages to our app.

1. In Visual Studio, open the tab for `MainWindow.xaml.cs`.
2. Add this code to your project:

```
// ***** GET TRANSLATABLE LANGUAGE CODES
private void GetLanguagesForTranslate()
{
    // Send request to get supported language codes
    string uri = String.Format(TEXT_TRANSLATION_API_ENDPOINT, "languages") + "&scope=translation";
    WebRequest WebRequest = WebRequest.Create(uri);
    WebRequest.Headers.Add("Accept-Language", "en");
    WebResponse response = null;
    // Read and parse the JSON response
    response = WebRequest.GetResponse();
    using (var reader = new StreamReader(response.GetResponseStream(), UnicodeEncoding.UTF8))
    {
        var result = JsonConvert.DeserializeObject<Dictionary<string, Dictionary<string,
Dictionary<string, string>>>>(reader.ReadToEnd());
        var languages = result["translation"];

        languageCodes = languages.Keys.ToArray();
        foreach (var kv in languages)
        {
            languageCodesAndTitles.Add(kv.Value["name"], kv.Key);
        }
    }
}
// NOTE:
// In the following sections, we'll add code below this.
```

The `GetLanguagesForTranslate()` method creates an HTTP GET request, and uses the `scope=translation` query string parameter to limit the scope of the request to supported languages for translation. The `Accept-Language` header with the value `en` is added so that the supported languages are returned in English.

The JSON response is parsed and converted to a dictionary. Then the language codes are added to the `languageCodes` member variable. The key/value pairs that contain the language codes and the friendly language names are looped through and added to the `languageCodesAndTitles` member variable. The drop-down menus in the form display the friendly names, but the codes are needed to request the translation.

Populate language drop-down menus

The user interface is defined using XAML, so you don't need to do much to set it up besides call `InitializeComponent()`. The one thing you need to do is add the friendly language names to the **Translate from** and **Translate to** drop-down menus. The `PopulateLanguageMenus()` method adds the names.

1. In Visual Studio, open the tab for `MainWindow.xaml.cs`.
2. Add this code to your project below the `GetLanguagesForTranslate()` method:


```
private void PopulateLanguageMenus()
{
    // Add option to automatically detect the source language
    FromLanguageComboBox.Items.Add("Detect");

    int count = languageCodesAndTitles.Count;
    foreach (string menuItem in languageCodesAndTitles.Keys)
    {
        FromLanguageComboBox.Items.Add(menuItem);
        ToLanguageComboBox.Items.Add(menuItem);
    }

    // Set default languages
    FromLanguageComboBox.SelectedItem = "Detect";
    ToLanguageComboBox.SelectedItem = "English";
}
// NOTE:
// In the following sections, we'll add code below this.
```

This method iterates over the `languageCodesAndTitles` dictionary and adds each key to both menus. After the menus are populated, the default from and to languages are set to **Detect** and **English** respectively.

TIP

Without a default selection for the menus, the user can click **Translate** without first choosing a "to" or "from" language. The defaults eliminate the need to deal with this problem.

Now that `MainWindow` has been initialized and the user interface created, this code won't run until the **Translate** button is clicked.

Detect language of source text

Now we're going to create method to detect the language of the source text (text entered into our text area) using the Translator Text API. The value returned by this request will be used in our translation request later.

1. In Visual Studio, open the tab for `MainWindow.xaml.cs`.
2. Add this code to your project below the `PopulateLanguageMenus()` method:

```
// ***** DETECT LANGUAGE OF TEXT TO BE TRANSLATED
private string DetectLanguage(string text)
{
    string detectUri = string.Format(TEXT_TRANSLATION_API_ENDPOINT , "detect");

    // Create request to Detect languages with Translator Text
    HttpRequest detectLanguageWebRequest = (HttpRequest)WebRequest.Create(detectUri);
    detectLanguageWebRequest.Headers.Add("Ocp-Apim-Subscription-Key", COGNITIVE_SERVICES_KEY);
    detectLanguageWebRequest.Headers.Add("Ocp-Apim-Subscription-Region", "westus");
    detectLanguageWebRequest.ContentType = "application/json; charset=utf-8";
    detectLanguageWebRequest.Method = "POST";

    // Send request
    var serializer = new System.Web.Script.Serialization.JavaScriptSerializer();
    string jsonText = serializer.Serialize(text);

    string body = "[{ \"Text\": \" + jsonText + \" }]";
    byte[] data = Encoding.UTF8.GetBytes(body);

    detectLanguageWebRequest.ContentLength = data.Length;

    using (var requestStream = detectLanguageWebRequest.GetRequestStream())
        requestStream.Write(data, 0, data.Length);

    HttpResponseMessage response = (HttpResponseMessage)detectLanguageWebRequest.GetResponse();

    // Read and parse JSON response
    var responseStream = response.GetResponseStream();
    var jsonString = new StreamReader(responseStream, Encoding.GetEncoding("utf-8")).ReadToEnd();
    dynamic jsonResponse = serializer.DeserializeObject(jsonString);

    // Fish out the detected language code
    var languageInfo = jsonResponse[0];
    if (languageInfo["score"] > (decimal)0.5)
    {
        DetectedLanguageLabel.Content = languageInfo["language"];
        return languageInfo["language"];
    }
    else
        return "Unable to confidently detect input language.";
}
// NOTE:
// In the following sections, we'll add code below this.
```

This method creates an HTTP `POST` request to the Detect resource. It takes a single argument, `text`, which is passed along as the body of the request. Later, when we create our translation request, the text entered into our UI will be passed to this method for language detection.

Additionally, this method evaluates the confidence score of the response. If the score is greater than `0.5`, then the detected language is displayed in our user interface.

Spell check the source text

Now we're going to create a method to spell check our source text using the Bing Spell Check API. Spell checking ensures that we'll get back accurate translations from Translator Text API. Any corrections to the source text are passed along in our translation request when the **Translate** button is clicked.

1. In Visual Studio, open the tab for `MainWindow.xaml.cs`.
2. Add this code to your project below the `DetectLanguage()` method:

```
// ***** CORRECT SPELLING OF TEXT TO BE TRANSLATED
private string CorrectSpelling(string text)
{
    string uri = BING_SPELL_CHECK_API_ENDPOINT + "?mode=spell&mkt=en-US";

    // Create a request to Bing Spell Check API
    HttpRequest spellCheckWebRequest = (HttpRequest)WebRequest.Create(uri);
    spellCheckWebRequest.Headers.Add("Ocp-Apim-Subscription-Key", COGNITIVE_SERVICES_KEY);
    spellCheckWebRequest.Method = "POST";
    spellCheckWebRequest.ContentType = "application/x-www-form-urlencoded"; // doesn't work without this

    // Create and send the request
    string body = "text=" + System.Web.HttpUtility.UrlEncode(text);
    byte[] data = Encoding.UTF8.GetBytes(body);
    spellCheckWebRequest.ContentLength = data.Length;
    using (var requestStream = spellCheckWebRequest.GetRequestStream())
        requestStream.Write(data, 0, data.Length);
    HttpResponse response = (HttpResponse)spellCheckWebRequest.GetResponse();

    // Read and parse the JSON response; get spelling corrections
    var serializer = new System.Web.Script.Serialization.JavaScriptSerializer();
    var responseStream = response.GetResponseStream();
    var jsonString = new StreamReader(responseStream, Encoding.GetEncoding("utf-8")).ReadToEnd();
    dynamic jsonResponse = serializer.DeserializeObject(jsonString);
    var flaggedTokens = jsonResponse["flaggedTokens"];

    // Construct sorted dictionary of corrections in reverse order (right to left)
    // This ensures that changes don't impact later indexes
    var corrections = new SortedDictionary<int, string[]>(Comparer<int>.Create((a, b) => b.CompareTo(a)));
    for (int i = 0; i < flaggedTokens.Length; i++)
    {
        var correction = flaggedTokens[i];
        var suggestion = correction["suggestions"][0]; // Consider only first suggestion
        if (suggestion["score"] > (decimal)0.7) // Take it only if highly confident
            corrections[(int)correction["offset"]] = new string[] // dict key = offset
                { correction["token"], suggestion["suggestion"] }; // dict value = {error, correction}
    }

    // Apply spelling corrections, in order, from right to left
    foreach (int i in corrections.Keys)
    {
        var oldtext = corrections[i][0];
        var newtext = corrections[i][1];

        // Apply capitalization from original text to correction - all caps or initial caps
        if (text.Substring(i, oldtext.Length).All(char.IsUpper)) newtext = newtext.ToUpper();
        else if (char.IsUpper(text[i])) newtext = newtext[0].ToString().ToUpper() + newtext.Substring(1);

        text = text.Substring(0, i) + newtext + text.Substring(i + oldtext.Length);
    }
    return text;
}
// NOTE:
// In the following sections, we'll add code below this.
```

Translate text on click

The last thing that we need to do is create a method that is invoked when the **Translate** button in our user interface is clicked.

1. In Visual Studio, open the tab for `MainWindow.xaml.cs`.
2. Add this code to your project below the `CorrectSpelling()` method and save:

```
// ***** PERFORM TRANSLATION ON BUTTON CLICK
```

```

private async void TranslateButton_Click(object sender, EventArgs e)
{
    string textToTranslate = TextToTranslate.Text.Trim();

    string fromLanguage = FromLanguageComboBox.SelectedValue.ToString();
    string fromLanguageCode;

    // auto-detect source language if requested
    if (fromLanguage == "Detect")
    {
        fromLanguageCode = DetectLanguage(textToTranslate);
        if (!languageCodes.Contains(fromLanguageCode))
        {
            MessageBox.Show("The source language could not be detected automatically " +
                "or is not supported for translation.", "Language detection failed",
                MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }
    }
    else
        fromLanguageCode = languageCodesAndTitles[fromLanguage];

    string toLanguageCode = languageCodesAndTitles[ToLanguageComboBox.SelectedValue.ToString()];

    // spell-check the source text if the source language is English
    if (fromLanguageCode == "en")
    {
        if (textToTranslate.StartsWith("-")) // don't spell check in this case
            textToTranslate = textToTranslate.Substring(1);
        else
        {
            textToTranslate = CorrectSpelling(textToTranslate);
            TextToTranslate.Text = textToTranslate; // put corrected text into input field
        }
    }

    // handle null operations: no text or same source/target languages
    if (textToTranslate == "" || fromLanguageCode == toLanguageCode)
    {
        TranslatedTextLabel.Content = textToTranslate;
        return;
    }

    // send HTTP request to perform the translation
    string endpoint = string.Format(TEXT_TRANSLATION_API_ENDPOINT, "translate");
    string uri = string.Format(endpoint + "&from={0}&to={1}", fromLanguageCode, toLanguageCode);

    System.Object[] body = new System.Object[] { new { Text = textToTranslate } };
    var requestBody = JsonConvert.SerializeObject(body);

    using (var client = new HttpClient())
    using (var request = new HttpRequestMessage())
    {
        request.Method = HttpMethod.Post;
        request.RequestUri = new Uri(uri);
        request.Content = new StringContent(requestBody, Encoding.UTF8, "application/json");
        request.Headers.Add("Ocp-Apim-Subscription-Key", COGNITIVE_SERVICES_KEY);
        request.Headers.Add("Ocp-Apim-Subscription-Region", "westus");
        request.Headers.Add("X-ClientTraceId", Guid.NewGuid().ToString());

        var response = await client.SendAsync(request);
        var responseBody = await response.Content.ReadAsStringAsync();

        var result = JsonConvert.DeserializeObject<List<Dictionary<string, List<Dictionary<string,
string>>>>>(responseBody);
        var translation = result[0]["translations"][0]["text"];

        // Update the translation field
        TranslatedTextLabel.Content = translation;
    }
}

```

```
}
```

The first step is to get the "from" and "to" languages, and the text the user entered into our form. If the source language is set to **Detect**, `DetectLanguage()` is called to determine the language of the source text. The text might be in a language that the Translator API doesn't support. In that case, display a message to inform the user, and return without translating the text.

If the source language is English (whether specified or detected), check the spelling of the text with `CorrectSpelling()` and apply any corrections. The corrected text is added back into the text area so that the user sees that a correction was made.

The code to translate text should look familiar: build the URI, create a request, send it, and parse the response. The JSON array may contain more than one object for translation, however, our app only requires one.

After a successful request, `TranslatedTextLabel.Content` is replaced with the `translation`, which updates the user interface to display the translated text.

Run your WPF app

That's it, you have a working translation app built using WPF. To run your app, click the **Start** button in Visual Studio.

Source code

Source code for this project is available on GitHub.

- [Explore source code](#)

Next steps

[Microsoft Translator Text API reference](#)

Tutorial: Build a Flask app with Azure Cognitive Services

12/10/2019 • 24 minutes to read • [Edit Online](#)

In this tutorial, you'll build a Flask web app that uses Azure Cognitive Services to translate text, analyze sentiment, and synthesize translated text into speech. Our focus is on the Python code and Flask routes that enable our application, however, we will help you out with the HTML and Javascript that pulls the app together. If you run into any issues let us know using the feedback button below.

Here's what this tutorial covers:

- Get Azure subscription keys
- Set up your development environment and install dependencies
- Create a Flask app
- Use the Translator Text API to translate text
- Use Text Analytics to analyze positive/negative sentiment of input text and translations
- Use Speech Services to convert translated text into synthesized speech
- Run your Flask app locally

TIP

If you'd like to skip ahead and see all the code at once, the entire sample, along with build instructions are available on [GitHub](#).

What is Flask?

Flask is a microframework for creating web applications. This means Flask provides you with tools, libraries, and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as substantive as a web-based calendar application or a commercial website.

For those of you who want to deep dive after this tutorial here are a few helpful links:

- [Flask documentation](#)
- [Flask for Dummies - A Beginner's Guide to Flask](#)

Prerequisites

Let's review the software and subscription keys that you'll need for this tutorial.

- [Python 3.5.2 or later](#)
- [Git tools](#)
- An IDE or text editor, such as [Visual Studio Code](#) or [Atom](#)
- [Chrome](#) or [Firefox](#)
- A **Translator Text** subscription key (Note that you aren't required to select a region.)
- A **Text Analytics** subscription key in the **West US** region.
- A **Speech Services** subscription key in the **West US** region.

Create an account and subscribe to resources

As previously mentioned, you're going to need three subscription keys for this tutorial. This means that you need to create a resource within your Azure account for:

- Translator Text
- Text Analytics
- Speech Services

Use [Create a Cognitive Services Account in the Azure portal](#) for step-by-step instructions to create resources.

IMPORTANT

For this tutorial, please create your resources in the West US region. If using a different region, you'll need to adjust the base URL in each of your Python files.

Set up your dev environment

Before you build your Flask web app, you'll need to create a working directory for your project and install a few Python packages.

Create a working directory

1. Open command line (Windows) or terminal (macOS/Linux). Then, create a working directory and sub directories for your project:

```
mkdir -p flask-cog-services/static/scripts && mkdir flask-cog-services/templates
```

2. Change to your project's working directory:

```
cd flask-cog-services
```

Create and activate your virtual environment with `virtualenv`

Let's create a virtual environment for our Flask app using `virtualenv`. Using a virtual environment ensures that you have a clean environment to work from.

1. In your working directory, run this command to create a virtual environment: **macOS/Linux:**

```
virtualenv venv --python=python3
```

We've explicitly declared that the virtual environment should use Python 3. This ensures that users with multiple Python installations are using the correct version.

Windows CMD / Windows Bash:

```
virtualenv venv
```

To keep things simple, we're naming your virtual environment `venv`.

2. The commands to activate your virtual environment will vary depending on your platform/shell:

PLATFORM	SHELL	COMMAND
macOS/Linux	bash/zsh	<code>source venv/bin/activate</code>

PLATFORM	SHELL	COMMAND
Windows	bash	<code>source venv/Scripts/activate</code>
	Command Line	<code>venv\Scripts\activate.bat</code>
	PowerShell	<code>venv\Scripts\Activate.ps1</code>

After running this command, your command line or terminal session should be prefaced with `venv`.

3. You can deactivate the session at any time by typing this into the command line or terminal: `deactivate`.

NOTE

Python has extensive documentation for creating and managing virtual environments, see [virtualenv](#).

Install requests

Requests is a popular module that is used to send HTTP 1.1 requests. There's no need to manually add query strings to your URLs, or to form-encode your POST data.

1. To install requests, run:

```
pip install requests
```

NOTE

If you'd like to learn more about requests, see [Requests: HTTP for Humans](#).

Install and configure Flask

Next we need to install Flask. Flask handles the routing for our web app, and allows us to make server-to-server calls that hide our subscription keys from the end user.

1. To install Flask, run:

```
pip install Flask
```

Let's make sure Flask was installed. Run:

```
flask --version
```

The version should be printed to terminal. Anything else means something went wrong.

2. To run the Flask app, you can either use the flask command or Python's `-m` switch with Flask. Before you can do that you need to tell your terminal which app to work with by exporting the `FLASK_APP` environment variable:

macOS/Linux:

```
export FLASK_APP=app.py
```

Windows:


```
set FLASK_APP=app.py
```

Create your Flask app

In this section, you're going to create a barebones Flask app that returns an HTML file when users hit the root of your app. Don't spend too much time trying to pick apart the code, we'll come back to update this file later.

What is a Flask route?

Let's take a minute to talk about "routes". Routing is used to bind a URL to a specific function. Flask uses route decorators to register functions to specific URLs. For example, when a user navigates to the root (/) of our web app, `index.html` is rendered.

```
@app.route('/')
def index():
    return render_template('index.html')
```

Let's take a look at one more example to hammer this home.

```
@app.route('/about')
def about():
    return render_template('about.html')
```

This code ensures that when a user navigates to `http://your-web-app.com/about` that the `about.html` file is rendered.

While these samples illustrate how to render html pages for a user, routes can also be used to call APIs when a button is pressed, or take any number of actions without having to navigate away from the homepage. You'll see this in action when you create routes for translation, sentiment, and speech synthesis.

Get started

1. Open the project in your IDE, then create a file named `app.py` in the root of your working directory. Next, copy this code into `app.py` and save:

```
from flask import Flask, render_template, url_for, jsonify, request

app = Flask(__name__)
app.config['JSON_AS_ASCII'] = False

@app.route('/')
def index():
    return render_template('index.html')
```

This code block tells the app to display `index.html` whenever a user navigates to the root of your web app (/).

2. Next, let's create the front-end for our web app. Create a file named `index.html` in the `templates` directory. Then copy this code into `templates/index.html`.

```

<!doctype html>
<html lang="en">
  <head>
    <!-- Required metadata tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="Translate and analyze text with Azure Cognitive Services.">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
    <title>Translate and analyze text with Azure Cognitive Services</title>
  </head>
  <body>
    <div class="container">
      <h1>Translate, synthesize, and analyze text with Azure</h1>
      <p>This simple web app uses Azure for text translation, text-to-speech conversion, and sentiment
analysis of input text and translations. Learn more about <a
href="https://docs.microsoft.com/azure/cognitive-services/">Azure Cognitive Services</a>.
      </p>
      <!-- HTML provided in the following sections goes here. -->

      <!-- End -->
    </div>

    <!-- Required Javascript for this tutorial -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkVYIK3UENzmM7KCKRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8j0t6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmY1" crossorigin="anonymous"></script>
    <script type = "text/javascript" src ="static/scripts/main.js"></script>
  </body>
</html>

```

3. Let's test the Flask app. From the terminal, run:

```
flask run
```

4. Open a browser and navigate to the URL provided. You should see your single page app. Press **Ctrl + c** to kill the app.

Translate text

Now that you have an idea of how a simple Flask app works, let's:

- Write some Python to call the Translator Text API and return a response
- Create a Flask route to call your Python code
- Update the HTML with an area for text input and translation, a language selector, and translate button
- Write Javascript that allows users to interact with your Flask app from the HTML

Call the Translator Text API

The first thing you need to do is write a function to call the Translator Text API. This function will take two arguments: `text_input` and `language_output`. This function is called whenever a user presses the translate button in your app. The text area in the HTML is sent as the `text_input`, and the language selection value in the HTML is sent as `language_output`.

1. Let's start by creating a file called `translate.py` in the root of your working directory.
2. Next, add this code to `translate.py`. This function takes two arguments: `text_input` and `language_output`.

```
import os, requests, uuid, json

# Don't forget to replace with your Cog Services subscription key!
# If you prefer to use environment variables, see Extra Credit for more info.
subscription_key = 'YOUR_TRANSLATOR_TEXT_SUBSCRIPTION_KEY'

# Don't forget to replace with your Cog Services location!
# Our Flask route will supply two arguments: text_input and language_output.
# When the translate text button is pressed in our Flask app, the Ajax request
# will grab these values from our web app, and use them in the request.
# See main.js for Ajax calls.
def get_translation(text_input, language_output):
    base_url = 'https://api.cognitive.microsofttranslator.com'
    path = '/translate?api-version=3.0'
    params = '&to=' + language_output
    constructed_url = base_url + path + params

    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key,
        'Ocp-Apim-Subscription-Region': 'location',
        'Content-type': 'application/json',
        'X-ClientTraceId': str(uuid.uuid4())
    }

    # You can pass more than one object in body.
    body = [{
        'text': text_input
    }]
    response = requests.post(constructed_url, headers=headers, json=body)
    return response.json()
```

3. Add your Translator Text subscription key and save.

Add a route to `app.py`

Next, you'll need to create a route in your Flask app that calls `translate.py`. This route will be called each time a user presses the translate button in your app.

For this app, your route is going to accept `POST` requests. This is because the function expects the text to translate and an output language for the translation.

Flask provides helper functions to help you parse and manage each request. In the code provided, `get_json()` returns the data from the `POST` request as JSON. Then using `data['text']` and `data['to']`, the text and output language values are passed to `get_translation()` function available from `translate.py`. The last step is to return the response as JSON, since you'll need to display this data in your web app.

In the following sections, you'll repeat this process as you create routes for sentiment analysis and speech synthesis.

1. Open `app.py` and locate the import statement at the top of `app.py` and add the following line:

```
import translate
```

Now our Flask app can use the method available via `translate.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/translate-text', methods=['POST'])
def translate_text():
    data = request.get_json()
    text_input = data['text']
    translation_output = data['to']
    response = translate.get_translation(text_input, translation_output)
    return jsonify(response)
```

Update `index.html`

Now that you have a function to translate text, and a route in your Flask app to call it, the next step is to start building the HTML for your app. The HTML below does a few things:

- Provides a text area where users can input text to translate.
- Includes a language selector.
- Includes HTML elements to render the detected language and confidence scores returned during translation.
- Provides a read-only text area where the translation output is displayed.
- Includes placeholders for sentiment analysis and speech synthesis code that you'll add to this file later in the tutorial.

Let's update `index.html`.

1. Open `index.html` and locate these code comments:

```
<!-- HTML provided in the following sections goes here. -->

<!-- End -->
```

2. Replace the code comments with this HTML block:

```

<div class="row">
  <div class="col">
    <form>
      <!-- Enter text to translate. -->
      <div class="form-group">
        <label for="text-to-translate"><strong>Enter the text you'd like to translate:</strong></label>
        <textarea class="form-control" id="text-to-translate" rows="5"></textarea>
      </div>
      <!-- Select output language. -->
      <div class="form-group">
        <label for="select-language"><strong>Translate to:</strong></label>
        <select class="form-control" id="select-language">
          <option value="ar">Arabic</option>
          <option value="ca">Catalan</option>
          <option value="zh-Hans">Chinese (Simplified)</option>
          <option value="zh-Hant">Chinese (Traditional)</option>
          <option value="hr">Croatian</option>
          <option value="en">English</option>
          <option value="fr">French</option>
          <option value="de">German</option>
          <option value="el">Greek</option>
          <option value="he">Hebrew</option>
          <option value="hi">Hindi</option>
          <option value="it">Italian</option>
          <option value="ja">Japanese</option>
          <option value="ko">Korean</option>
          <option value="pt">Portuguese</option>
          <option value="ru">Russian</option>
          <option value="es">Spanish</option>
          <option value="th">Thai</option>
          <option value="tr">Turkish</option>
          <option value="vi">Vietnamese</option>
        </select>
      </div>
      <button type="submit" class="btn btn-primary mb-2" id="translate">Translate text</button><br>
      <div id="detected-language" style="display: none">
        <strong>Detected language:</strong> <span id="detected-language-result"></span><br />
        <strong>Detection confidence:</strong> <span id="confidence"></span><br /><br />
      </div>

      <!-- Start sentiment code-->

      <!-- End sentiment code -->

    </form>
  </div>
  <div class="col">
    <!-- Translated text returned by the Translate API is rendered here. -->
    <form>
      <div class="form-group" id="translator-text-response">
        <label for="translation-result"><strong>Translated text:</strong></label>
        <textarea readonly class="form-control" id="translation-result" rows="5"></textarea>
      </div>

      <!-- Start voice font selection code -->

      <!-- End voice font selection code -->

    </form>

    <!-- Add Speech Synthesis button and audio element -->

    <!-- End Speech Synthesis button -->

  </div>
</div>

```

The next step is to write some Javascript. This is the bridge between your HTML and Flask route.

Create `main.js`

The `main.js` file is the bridge between your HTML and Flask route. Your app will use a combination of jQuery, Ajax, and XMLHttpRequest to render content, and make `POST` requests to your Flask routes.

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the contents of the text area and the language selector are assigned to variables, and then passed along in the request to `translate-text`.

The code then iterates through the response, and updates the HTML with the translation, detected language, and confidence score.

1. From your IDE, create a file named `main.js` in the `static/scripts` directory.
2. Copy this code into `static/scripts/main.js`:

```
//Initiate jQuery on load.
$(function() {
  //Translate text with flask route
  $("#translate").on("click", function(e) {
    e.preventDefault();
    var translateVal = document.getElementById("text-to-translate").value;
    var languageVal = document.getElementById("select-language").value;
    var translateRequest = { 'text': translateVal, 'to': languageVal }

    if (translateVal !== "") {
      $.ajax({
        url: '/translate-text',
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        dataType: 'json',
        data: JSON.stringify(translateRequest),
        success: function(data) {
          for (var i = 0; i < data.length; i++) {
            document.getElementById("translation-result").textContent = data[i].translations[0].text;
            document.getElementById("detected-language-result").textContent =
data[i].detectedLanguage.language;
            if (document.getElementById("detected-language-result").textContent !== ""){
              document.getElementById("detected-language").style.display = "block";
            }
            document.getElementById("confidence").textContent = data[i].detectedLanguage.score;
          }
        }
      });
    }
  });
});
// In the following sections, you'll add code for sentiment analysis and
// speech synthesis here.
})
```

Test translation

Let's test translation in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. If it doesn't work, make sure that you've added your subscription key.

TIP

If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

Press **CTRL + c** to kill the app, then head to the next section.

Analyze sentiment

The [Text Analytics API](#) can be used to perform sentiment analysis, extract key phrases from text, or detect the source language. In this app, we're going to use sentiment analysis to determine if the provided text is positive, neutral, or negative. The API returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment, and scores close to 0 indicate negative sentiment.

In this section, you're going to do a few things:

- Write some Python to call the Text Analytics API to perform sentiment analysis and return a response
- Create a Flask route to call your Python code
- Update the HTML with an area for sentiment scores, and a button to perform analysis
- Write Javascript that allows users to interact with your Flask app from the HTML

Call the Text Analytics API

Let's write a function to call the Text Analytics API. This function will take four arguments: `input_text`, `input_language`, `output_text`, and `output_language`. This function is called whenever a user presses the run sentiment analysis button in your app. Data provided by the user from the text area and language selector, as well as the detected language and translation output are provided with each request. The response object includes sentiment scores for the source and translation. In the following sections, you're going to write some Javascript to parse the response and use it in your app. For now, let's focus on call the Text Analytics API.

1. Let's create a file called `sentiment.py` in the root of your working directory.
2. Next, add this code to `sentiment.py`.

```

import os, requests, uuid, json

# Don't forget to replace with your Cog Services subscription key!
subscription_key = 'YOUR_TEXT_ANALYTICS_SUBSCRIPTION_KEY'

# Our Flask route will supply four arguments: input_text, input_language,
# output_text, output_language.
# When the run sentiment analysis button is pressed in our Flask app,
# the Ajax request will grab these values from our web app, and use them
# in the request. See main.js for Ajax calls.

def get_sentiment(input_text, input_language, output_text, output_language):
    base_url = 'https://westus.api.cognitive.microsoft.com/text/analytics'
    path = '/v2.0/sentiment'
    constructed_url = base_url + path

    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key,
        'Content-type': 'application/json',
        'X-ClientTraceId': str(uuid.uuid4())
    }

    # You can pass more than one object in body.
    body = {
        'documents': [
            {
                'language': input_language,
                'id': '1',
                'text': input_text
            },
            {
                'language': output_language,
                'id': '2',
                'text': output_text
            }
        ]
    }
    response = requests.post(constructed_url, headers=headers, json=body)
    return response.json()

```

3. Add your Text Analytics subscription key and save.

Add a route to `app.py`

Let's create a route in your Flask app that calls `sentiment.py`. This route will be called each time a user presses the run sentiment analysis button in your app. Like the route for translation, this route is going to accept `POST` requests since the function expects arguments.

1. Open `app.py` and locate the import statement at the top of `app.py` and update it:

```
import translate, sentiment
```

Now our Flask app can use the method available via `sentiment.py`.

2. Copy this code to the end of `app.py` and save:


```
@app.route('/sentiment-analysis', methods=['POST'])
def sentiment_analysis():
    data = request.get_json()
    input_text = data['inputText']
    input_lang = data['inputLanguage']
    output_text = data['outputText']
    output_lang = data['outputLanguage']
    response = sentiment.get_sentiment(input_text, input_lang, output_text, output_lang)
    return jsonify(response)
```

Update `index.html`

Now that you have a function to run sentiment analysis, and a route in your Flask app to call it, the next step is to start writing the HTML for your app. The HTML below does a few things:

- Adds a button to your app to run sentiment analysis
- Adds an element that explains sentiment scoring
- Adds an element to display the sentiment scores

1. Open `index.html` and locate these code comments:

```
<!-- Start sentiment code-->

<!-- End sentiment code -->
```

2. Replace the code comments with this HTML block:

```
<button type="submit" class="btn btn-primary mb-2" id="sentiment-analysis">Run sentiment
analysis</button><br>
<div id="sentiment" style="display: none">
  <p>Sentiment scores are provided on a 1 point scale. The closer the sentiment score is to 1,
  indicates positive sentiment. The closer it is to 0, indicates negative sentiment.</p>
  <strong>Sentiment score for input:</strong> <span id="input-sentiment"></span><br />
  <strong>Sentiment score for translation:</strong> <span id="translation-sentiment"></span>
</div>
```

Update `main.js`

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the contents of the text area and the language selector are assigned to variables, and then passed along in the request to the `sentiment-analysis` route.

The code then iterates through the response, and updates the HTML with the sentiment scores.

1. From your IDE, create a file named `main.js` in the `static` directory.
2. Copy this code into `static/scripts/main.js`:

```
//Run sentiment analysis on input and translation.
$("#sentiment-analysis").on("click", function(e) {
    e.preventDefault();
    var inputText = document.getElementById("text-to-translate").value;
    var inputLanguage = document.getElementById("detected-language-result").innerHTML;
    var outputText = document.getElementById("translation-result").value;
    var outputLanguage = document.getElementById("select-language").value;

    var sentimentRequest = { "inputText": inputText, "inputLanguage": inputLanguage, "outputText":
outputText, "outputLanguage": outputLanguage };

    if (inputText !== "") {
        $.ajax({
            url: "/sentiment-analysis",
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            dataType: "json",
            data: JSON.stringify(sentimentRequest),
            success: function(data) {
                for (var i = 0; i < data.documents.length; i++) {
                    if (typeof data.documents[i] !== "undefined"){
                        if (data.documents[i].id === "1") {
                            document.getElementById("input-sentiment").textContent = data.documents[i].score;
                        }
                        if (data.documents[i].id === "2") {
                            document.getElementById("translation-sentiment").textContent = data.documents[i].score;
                        }
                    }
                }
                for (var i = 0; i < data.errors.length; i++) {
                    if (typeof data.errors[i] !== "undefined"){
                        if (data.errors[i].id === "1") {
                            document.getElementById("input-sentiment").textContent = data.errors[i].message;
                        }
                        if (data.errors[i].id === "2") {
                            document.getElementById("translation-sentiment").textContent = data.errors[i].message;
                        }
                    }
                }
                if (document.getElementById("input-sentiment").textContent !== '' &&
document.getElementById("translation-sentiment").textContent !== ""){
                    document.getElementById("sentiment").style.display = "block";
                }
            }
        });
    }
});

// In the next section, you'll add code for speech synthesis here.
```

Test sentiment analysis

Let's test sentiment analysis in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. Next, press the run sentiment analysis button. You should see two scores. If it doesn't work, make sure that you've added your subscription key.

TIP

If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

Press **CTRL + c** to kill the app, then head to the next section.

Convert text-to-speech

The [Text-to-speech API](#) enables your app to convert text into natural human-like synthesized speech. The service supports standard, neural, and custom voices. Our sample app uses a handful of the available voices, for a full list, see [supported languages](#).

In this section, you're going to do a few things:

- Write some Python to convert text-to-speech with the Text-to-speech API
- Create a Flask route to call your Python code
- Update the HTML with a button to convert text-to-speech, and an element for audio playback
- Write Javascript that allows users to interact with your Flask app

Call the Text-to-Speech API

Let's write a function to convert text-to-speech. This function will take two arguments: `input_text` and `voice_font`. This function is called whenever a user presses the convert text-to-speech button in your app. `input_text` is the translation output returned by the call to translate text, `voice_font` is the value from the voice font selector in the HTML.

1. Let's create a file called `synthesize.py` in the root of your working directory.
2. Next, add this code to `synthesize.py`.

```

import os, requests, time
from xml.etree import ElementTree

class TextToSpeech(object):
    def __init__(self, input_text, voice_font):
        subscription_key = 'YOUR_SPEECH_SERVICES_SUBSCRIPTION_KEY'
        self.subscription_key = subscription_key
        self.input_text = input_text
        self.voice_font = voice_font
        self.timestr = time.strftime('%Y%m%d-%H%M')
        self.access_token = None

    # This function performs the token exchange.
    def get_token(self):
        fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
        headers = {
            'Ocp-Apim-Subscription-Key': self.subscription_key
        }
        response = requests.post(fetch_token_url, headers=headers)
        self.access_token = str(response.text)

    # This function calls the TTS endpoint with the access token.
    def save_audio(self):
        base_url = 'https://westus.tts.speech.microsoft.com/'
        path = 'cognitiveservices/v1'
        constructed_url = base_url + path
        headers = {
            'Authorization': 'Bearer ' + self.access_token,
            'Content-Type': 'application/ssml+xml',
            'X-Microsoft-OutputFormat': 'riff-24khz-16bit-mono-pcm',
            'User-Agent': 'YOUR_RESOURCE_NAME',
        }
        # Build the SSML request with ElementTree
        xml_body = ElementTree.Element('speak', version='1.0')
        xml_body.set('{http://www.w3.org/XML/1998/namespace}lang', 'en-us')
        voice = ElementTree.SubElement(xml_body, 'voice')
        voice.set('{http://www.w3.org/XML/1998/namespace}lang', 'en-US')
        voice.set('name', 'Microsoft Server Speech Text to Speech Voice {}'.format(self.voice_font))
        voice.text = self.input_text
        # The body must be encoded as UTF-8 to handle non-ascii characters.
        body = ElementTree.tostring(xml_body, encoding="utf-8")

        #Send the request
        response = requests.post(constructed_url, headers=headers, data=body)

        # Write the response as a wav file for playback. The file is located
        # in the same directory where this sample is run.
        return response.content

```

3. Add your Speech Services subscription key and save.

Add a route to `app.py`

Let's create a route in your Flask app that calls `synthesize.py`. This route will be called each time a user presses the convert text-to-speech button in your app. Like the routes for translation and sentiment analysis, this route is going to accept `POST` requests since the function expects two arguments: the text to synthesize, and the voice font for playback.

1. Open `app.py` and locate the import statement at the top of `app.py` and update it:

```
import translate, sentiment, synthesize
```

Now our Flask app can use the method available via `synthesize.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/text-to-speech', methods=['POST'])
def text_to_speech():
    data = request.get_json()
    text_input = data['text']
    voice_font = data['voice']
    tts = synthesizer.TextToSpeech(text_input, voice_font)
    tts.get_token()
    audio_response = tts.save_audio()
    return audio_response
```

Update `index.html`

Now that you have a function to convert text-to-speech, and a route in your Flask app to call it, the next step is to start writing the HTML for your app. The HTML below does a few things:

- Provides a voice selection drop-down
- Adds a button to convert text-to-speech
- Adds an audio element, which is used to play back the synthesized speech

1. Open `index.html` and locate these code comments:

```
<!-- Start voice font selection code -->

<!-- End voice font selection code -->
```

2. Replace the code comments with this HTML block:

```

<div class="form-group">
  <label for="select-voice"><strong>Select voice font:</strong></label>
  <select class="form-control" id="select-voice">
    <option value="(ar-SA, Naayf)">Arabic | Male | Naayf</option>
    <option value="(ca-ES, HerenaRUS)">Catalan | Female | HerenaRUS</option>
    <option value="(zh-CN, HuihuiRUS)">Chinese (Mainland) | Female | HuihuiRUS</option>
    <option value="(zh-CN, Kangkang, Apollo)">Chinese (Mainland) | Male | Kangkang, Apollo</option>
    <option value="(zh-HK, Tracy, Apollo)">Chinese (Hong Kong)| Female | Tracy, Apollo</option>
    <option value="(zh-HK, Danny, Apollo)">Chinese (Hong Kong) | Male | Danny, Apollo</option>
    <option value="(zh-TW, Yating, Apollo)">Chinese (Taiwan)| Female | Yating, Apollo</option>
    <option value="(zh-TW, Zhiwei, Apollo)">Chinese (Taiwan) | Male | Zhiwei, Apollo</option>
    <option value="(hr-HR, Matej)">Croatian | Male | Matej</option>
    <option value="(en-US, Jessa24kRUS)">English (US) | Female | Jessa24kRUS</option>
    <option value="(en-US, Guy24kRUS)">English (US) | Male | Guy24kRUS</option>
    <option value="(en-IE, Sean)">English (IE) | Male | Sean</option>
    <option value="(fr-FR, Julie, Apollo)">French | Female | Julie, Apollo</option>
    <option value="(fr-FR, HortenseRUS)">French | Female | Julie, HortenseRUS</option>
    <option value="(fr-FR, Paul, Apollo)">French | Male | Paul, Apollo</option>
    <option value="(de-DE, Hedda)">German | Female | Hedda</option>
    <option value="(de-DE, HeddaRUS)">German | Female | HeddaRUS</option>
    <option value="(de-DE, Stefan, Apollo)">German | Male | Apollo</option>
    <option value="(el-GR, Stefanos)">Greek | Male | Stefanos</option>
    <option value="(he-IL, Asaf)">Hebrew (Isreal) | Male | Asaf</option>
    <option value="(hi-IN, Kalpana, Apollo)">Hindi | Female | Kalpana, Apollo</option>
    <option value="(hi-IN, Hemant)">Hindi | Male | Hemant</option>
    <option value="(it-IT, LuciaRUS)">Italian | Female | LuciaRUS</option>
    <option value="(it-IT, Cosimo, Apollo)">Italian | Male | Cosimo, Apollo</option>
    <option value="(ja-JP, Ichiro, Apollo)">Japanese | Male | Ichiro</option>
    <option value="(ja-JP, HarukaRUS)">Japanese | Female | HarukaRUS</option>
    <option value="(ko-KR, HeamiRUS)">Korean | Female | Haemi</option>
    <option value="(pt-BR, HeloisaRUS)">Portuguese (Brazil) | Female | HeloisaRUS</option>
    <option value="(pt-BR, Daniel, Apollo)">Portuguese (Brazil) | Male | Daniel, Apollo</option>
    <option value="(pt-PT, HeliaRUS)">Portuguese (Portugal) | Female | HeliaRUS</option>
    <option value="(ru-RU, Irina, Apollo)">Russian | Female | Irina, Apollo</option>
    <option value="(ru-RU, Pavel, Apollo)">Russian | Male | Pavel, Apollo</option>
    <option value="(ru-RU, EkaterinaRUS)">Russian | Female | EkaterinaRUS</option>
    <option value="(es-ES, Laura, Apollo)">Spanish | Female | Laura, Apollo</option>
    <option value="(es-ES, HelenaRUS)">Spanish | Female | HelenaRUS</option>
    <option value="(es-ES, Pablo, Apollo)">Spanish | Male | Pablo, Apollo</option>
    <option value="(th-TH, Pattara)">Thai | Male | Pattara</option>
    <option value="(tr-TR, SedaRUS)">Turkish | Female | SedaRUS</option>
    <option value="(vi-VN, An)">Vietnamese | Male | An</option>
  </select>
</div>

```

3. Next, locate these code comments:

```

<!-- Add Speech Synthesis button and audio element -->

<!-- End Speech Synthesis button -->

```

4. Replace the code comments with this HTML block:

```

<button type="submit" class="btn btn-primary mb-2" id="text-to-speech">Convert text-to-speech</button>
<div id="audio-playback">
  <audio id="audio" controls>
    <source id="audio-source" type="audio/mpeg" />
  </audio>
</div>

```

5. Make sure to save your work.

Update `main.js`

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the translation and the voice font are assigned to variables, and then passed along in the request to the `text-to-speech` route.

The code then iterates through the response, and updates the HTML with the sentiment scores.

1. From your IDE, create a file named `main.js` in the `static` directory.
2. Copy this code into `static/scripts/main.js`:

```
// Convert text-to-speech
$("#text-to-speech").on("click", function(e) {
  e.preventDefault();
  var ttsInput = document.getElementById("translation-result").value;
  var ttsVoice = document.getElementById("select-voice").value;
  var ttsRequest = { 'text': ttsInput, 'voice': ttsVoice }

  var xhr = new XMLHttpRequest();
  xhr.open("post", "/text-to-speech", true);
  xhr.setRequestHeader("Content-Type", "application/json");
  xhr.responseType = "blob";
  xhr.onload = function(evt){
    if (xhr.status === 200) {
      audioBlob = new Blob([xhr.response], {type: "audio/mpeg"});
      audioURL = URL.createObjectURL(audioBlob);
      if (audioURL.length > 5){
        var audio = document.getElementById("audio");
        var source = document.getElementById("audio-source");
        source.src = audioURL;
        audio.load();
        audio.play();
      }else{
        console.log("An error occurred getting and playing the audio.")
      }
    }
  }
  xhr.send(JSON.stringify(ttsRequest));
});
// Code for automatic language selection goes here.
```

3. You're almost done. The last thing you're going to do is add some code to `main.js` to automatically select a voice font based on the language selected for translation. Add this code block to `main.js`:

```
// Automatic voice font selection based on translation output.
$('select[id="select-language"]').change(function(e) {
  if ($(this).val() == "ar"){
    document.getElementById("select-voice").value = "(ar-SA, Naayf)";
  }
  if ($(this).val() == "ca"){
    document.getElementById("select-voice").value = "(ca-ES, HerenaRUS)";
  }
  if ($(this).val() == "zh-Hans"){
    document.getElementById("select-voice").value = "(zh-HK, Tracy, Apollo)";
  }
  if ($(this).val() == "zh-Hant"){
    document.getElementById("select-voice").value = "(zh-HK, Tracy, Apollo)";
  }
  if ($(this).val() == "hr"){
    document.getElementById("select-voice").value = "(hr-HR, Matej)";
  }
  if ($(this).val() == "en"){
    document.getElementById("select-voice").value = "(en-US, Jessa24kRUS)";
  }
  if ($(this).val() == "fr"){
    document.getElementById("select-voice").value = "(fr-FR, HortenseRUS)";
  }
  if ($(this).val() == "de"){
    document.getElementById("select-voice").value = "(de-DE, HeddaRUS)";
  }
  if ($(this).val() == "el"){
    document.getElementById("select-voice").value = "(el-GR, Stefanos)";
  }
  if ($(this).val() == "he"){
    document.getElementById("select-voice").value = "(he-IL, Asaf)";
  }
  if ($(this).val() == "hi"){
    document.getElementById("select-voice").value = "(hi-IN, Kalpana, Apollo)";
  }
  if ($(this).val() == "it"){
    document.getElementById("select-voice").value = "(it-IT, LuciaRUS)";
  }
  if ($(this).val() == "ja"){
    document.getElementById("select-voice").value = "(ja-JP, HarukaRUS)";
  }
  if ($(this).val() == "ko"){
    document.getElementById("select-voice").value = "(ko-KR, HeamiRUS)";
  }
  if ($(this).val() == "pt"){
    document.getElementById("select-voice").value = "(pt-BR, HeloisaRUS)";
  }
  if ($(this).val() == "ru"){
    document.getElementById("select-voice").value = "(ru-RU, EkaterinaRUS)";
  }
  if ($(this).val() == "es"){
    document.getElementById("select-voice").value = "(es-ES, HelenaRUS)";
  }
  if ($(this).val() == "th"){
    document.getElementById("select-voice").value = "(th-TH, Pattara)";
  }
  if ($(this).val() == "tr"){
    document.getElementById("select-voice").value = "(tr-TR, SedaRUS)";
  }
  if ($(this).val() == "vi"){
    document.getElementById("select-voice").value = "(vi-VN, An)";
  }
});
```

Test your app

Let's test speech synthesis in the app.


```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. Next, select a voice, then press the convert text-to-speech button. the translation should be played back as synthesized speech. If it doesn't work, make sure that you've added your subscription key.

TIP

If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

That's it, you have a working app that performs translations, analyzes sentiment, and synthesized speech. Press **CTRL + c** to kill the app. Be sure to check out the other [Azure Cognitive Services](#).

Get the source code

The source code for this project is available on [GitHub](#).

Next steps

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

Customize your text translations

9/24/2019 • 2 minutes to read • [Edit Online](#)

The Microsoft Custom Translator is feature of the Microsoft Translator service, which allows users to customize Microsoft Translator's advanced neural machine translation when translating text using the Translator Text API (version 3 only).

The feature can also be used to customize speech translation when used with [Cognitive Services Speech](#).

Custom Translator

With Custom Translator, you can build neural translation systems that understand the terminology used in your own business and industry. The customized translation system will then integrate into existing applications, workflows, and websites.

How does it work?

Use your previously translated documents (leaflets, webpages, documentation, etc.) to build a translation system that reflects your domain-specific terminology and style, better than a standard translation system. Users can upload TMX, XLIFF, TXT, DOCX, and XLSX documents.

The system also accepts data that is parallel at the document level but is not yet aligned at the sentence level. If users have access to versions of the same content in multiple languages but in separate documents Custom Translator will be able to automatically match sentences across documents. The system can also use monolingual data in either or both languages to complement the parallel training data to improve the translations.

The customized system is then available through a regular call to the Microsoft Translator Text API using the category parameter.

Given the appropriate type and amount of training data it is not uncommon to expect gains between 5 and 10, or even more BLEU points on translation quality by using Custom Translator.

More details about the various levels of customization based on available data can be found in the [Custom Translator User Guide](#).

Microsoft Translator Hub

NOTE

The legacy Microsoft Translator Hub will be retired on May 17, 2019. [View important migration information and dates](#).

Custom Translator versus Hub

	HUB	CUSTOM TRANSLATOR
Customization Feature Status	General Availability	General Availability
Text API version	V2 only	V3 only
SMT customization	Yes	No

	HUB	CUSTOM TRANSLATOR
NMT customization	No	Yes
New unified Speech services customization	No	Yes
No Trace	Yes	Yes

Collaborative Translations Framework

NOTE

As of February 1, 2018, `AddTranslation()` and `AddTranslationArray()` are no longer available for use with the Translator Text API V2.0. These methods will fail and nothing will be written. The Translator Text API V3.0 does not support these methods.

Next steps

[Set up a customized language system using Custom Translator](#)

How the Translator Text API counts characters

11/8/2019 • 2 minutes to read • [Edit Online](#)

The Translator Text API counts every Unicode code point of input text as a character. Each translation of a text to a language counts as a separate translation, even if the request was made in a single API call translating to multiple languages. The length of the response does not matter.

What counts is:

- Text passed to the Translator Text API in the body of the request
 - `Text` when using the Translate, Transliterate, and Dictionary Lookup methods
 - `Text` and `Translation` when using the Dictionary Examples method
- All markup: HTML, XML tags, etc. within the text field of the request body. JSON notation used to build the request (for instance "Text:") is not counted.
- An individual letter
- Punctuation
- A space, tab, markup, and any kind of white space character
- Every code point defined in Unicode
- A repeated translation, even if you have translated the same text previously

For scripts based on ideograms such as Chinese and Japanese Kanji, the Translator Text API will still count the number of Unicode code points, one character per ideogram. Exception: Unicode surrogates count as two characters.

The number of requests, words, bytes, or sentences is irrelevant in the character count.

Calls to the Detect and BreakSentence methods are not counted in the character consumption. However, we do expect that the calls to the Detect and BreakSentence methods are in a reasonable proportion to the use of other functions that are counted. If the number of Detect or BreakSentence calls you make exceeds the number of other counted methods by 100 times, Microsoft reserves the right to restrict your use of the Detect and BreakSentence methods.

More information about character counts is in the [Microsoft Translator FAQ](#).

How to sign up for the Translator Text API

11/8/2019 • 2 minutes to read • [Edit Online](#)

Sign in to the Azure portal

- Don't have an account? You can create a [free account](#) to experiment at no charge.
- Already have an account? [Sign in](#)

Create a subscription to the Translator Text API

After you sign in to the portal, you can create a subscription to the Translator Text API as follows:

1. Select + **Create a resource**.
2. In the **Search the Marketplace** search box, enter **Translator Text** and then select it from the results.
3. Select **Create** to define details for the subscription.
4. From the **Pricing tier** list, select the pricing tier that best fits your needs.
 - a. Each subscription has a free tier. The free tier has the same features and functionalities as the paid plans and doesn't expire.
 - b. You can have only one free subscription for your account.
5. Select **Create** to finish creating the subscription.

Authentication key

When you sign up for Translator Text, you get a personalized access key unique to your subscription. This key is required on each call to the Translator Text API.

1. Retrieve your authentication key by first selecting the appropriate subscription.
2. Select **Keys** in the **Resource Management** section of your subscription's details.
3. Copy either of the keys listed for your subscription.

Learn, test, and get support

- [Code examples on GitHub](#)
- [Microsoft Translator Support Forum](#)

Microsoft Translator will generally let your first couple of requests pass before it has verified the subscription account status. If the first few Microsoft Translator API requests succeed then the calls fail, the error response will indicate the problem. Please log the API response so you can see the reason.

Pricing options

- [Translator Text API](#)

Customization

Use Custom Translator to customize your translations and create a translation system tuned to your own terminology and style, starting from generic Microsoft Translator neural machine translation systems. [Learn more](#)

Additional resources

- [Get Started with Azure \(3-minute video\)](#)
- [How to Pay with an Invoice](#)

Translator Text API V2 to V3 Migration

11/8/2019 • 5 minutes to read • [Edit Online](#)

NOTE

V2 was deprecated on April 30, 2018. Please migrate your applications to V3 in order to take advantage of new functionality available exclusively in V3.

The Microsoft Translator Hub will be retired on May 17, 2019. [View important migration information and dates.](#)

The Microsoft Translator team has released Version 3 (V3) of the Translator Text API. This release includes new features, deprecated methods and a new format for sending to, and receiving data from the Microsoft Translator Service. This document provides information for changing applications to use V3.

The end of this document contains helpful links for you to learn more.

Summary of features

- No Trace - In V3 No-Trace applies to all pricing tiers in the Azure portal. This feature means that no text submitted to the V3 API, will be saved by Microsoft.
- JSON - XML is replaced by JSON. All data sent to the service and received from the service is in JSON format.
- Multiple target languages in a single request - The Translate method accepts multiple 'to' languages for translation in a single request. For example, a single request can be 'from' English and 'to' German, Spanish and Japanese, or any other group of languages.
- Bilingual dictionary - A bilingual dictionary method has been added to the API. This method includes 'lookup' and 'examples'.
- Transliterate - A transliterate method has been added to the API. This method will convert words and sentences in one script (E.g. Arabic) into another script (E.g. Latin).
- Languages - A new 'languages' method delivers language information, in JSON format, for use with the 'translate', 'dictionary', and 'transliterate' methods.
- New to Translate - New capabilities have been added to the 'translate' method to support some of the features that were in the V2 API as separate methods. An example is TranslateArray.
- Speak method - Text to speech functionality is no longer supported in the Microsoft Translator API. Text to speech functionality is available in [Microsoft Speech Service](#).

The following list of V2 and V3 methods identifies the V3 methods and APIs that will provide the functionality that came with V2.

V2 API METHOD	V3 API COMPATIBILITY
<code>Translate</code>	Translate
<code>TranslateArray</code>	Translate
<code>GetLanguageNames</code>	Languages
<code>GetLanguagesForTranslate</code>	Languages

V2 API METHOD	V3 API COMPATIBILITY
<code>GetLanguagesForSpeak</code>	Microsoft Speech Service
<code>Speak</code>	Microsoft Speech Service
<code>Detect</code>	Detect
<code>DetectArray</code>	Detect
<code>AddTranslation</code>	Feature is no longer supported
<code>AddTranslationArray</code>	Feature is no longer supported
<code>BreakSentences</code>	BreakSentence
<code>GetTranslations</code>	Feature is no longer supported
<code>GetTranslationsArray</code>	Feature is no longer supported

Move to JSON format

Microsoft Translator Text Translation V2 accepted and returned data in XML format. In V3 all data sent and received using the API is in JSON format. XML will no longer be accepted or returned in V3.

This change will affect several aspects of an application written for the V2 Text Translation API. As an example: The Languages API returns language information for text translation, transliteration, and the two dictionary methods. You can request all language information for all methods in one call or request them individually.

The languages method does not require authentication; by clicking on the following link you can see all the language information for V3 in JSON:

<https://api.cognitive.microsofttranslator.com/languages?api-version=3.0&scope=translation,dictionary,transliteration>

Authentication Key

The authentication key you are using for V2 will be accepted for V3. You will not need to get a new subscription. You will be able to mix V2 and V3 in your apps during the yearlong migration period, making it easier for you to release new versions while you are still migrating from V2-XML to V3-JSON.

Pricing Model

Microsoft Translator V3 is priced in the same way V2 was priced; per character, including spaces. The new features in V3 make some changes in what characters are counted for billing.

V3 METHOD	CHARACTERS COUNTED FOR BILLING
<code>Languages</code>	No characters submitted, none counted, no charge.

V3 METHOD	CHARACTERS COUNTED FOR BILLING
Translate	Count is based on how many characters are submitted for translation, and how many languages the characters are translated into. 50 characters submitted, and 5 languages requested will be 50x5.
Transliterate	Number of characters submitted for transliteration are counted.
Dictionary lookup & example	Number of characters submitted for Dictionary lookup and examples are counted.
BreakSentence	No Charge.
Detect	No Charge.

V3 End Points

Global

- api.cognitive.microsofttranslator.com

V3 API text translations methods

[Languages](#)

[Translate](#)

[Transliterate](#)

[BreakSentence](#)

[Detect](#)

[Dictionary/lookup](#)

[Dictionary/example](#)

Compatibility and customization

NOTE

The Microsoft Translator Hub will be retired on May 17, 2019. [View important migration information and dates.](#)

Microsoft Translator V3 uses neural machine translation by default. As such, it cannot be used with the Microsoft Translator Hub. The Translator Hub only supports legacy statistical machine translation. Customization for neural translation is now available using the Custom Translator. [Learn more about customizing neural machine translation](#)

Neural translation with the V3 text API does not support the use of standard categories (SMT, speech, tech, generalnn).

	ENDPOINT	GDPR PROCESSOR COMPLIANCE	USE TRANSLATOR HUB	USE CUSTOM TRANSLATOR (PREVIEW)
Translator Text API Version 2	api.microsofttranslator.com	No	Yes	No
Translator Text API Version 3	api.cognitive.microsofttranslator.com	Yes	No	Yes

Translator Text API Version 3

- Is generally available and fully supported.
- Is GDPR compliant as a processor and satisfies all ISO 20001 and 20018 as well as SOC 3 certification requirements.
- Allows you to invoke the neural network translation systems you have customized with Custom Translator (Preview), the new Translator NMT customization feature.
- Does not provide access to custom translation systems created using the Microsoft Translator Hub.

You are using Version 3 of the Translator Text API If you are using the [api.cognitive.microsofttranslator.com](#) endpoint.

Translator Text API Version 2

- Does not satisfy all ISO 20001,20018 and SOC 3 certification requirements.
- Does not allow you to invoke the neural network translation systems you have customized with the Translator customization feature.
- Provides access to custom translation systems created using the Microsoft Translator Hub.
- You are using Version 2 of the Translator Text API If you are using the [api.microsofttranslator.com](#) endpoint.

No version of the Translator API creates a record of your translations. Your translations are never shared with anyone. More information on the [Translator No-Trace](#) webpage.

Links

- [Microsoft Privacy Policy](#)
- [Microsoft Azure Legal Information](#)
- [Online Services Terms](#)

Next steps

[View V3.0 Documentation](#)

Add profanity filtering with the Translator Text API

11/8/2019 • 2 minutes to read • [Edit Online](#)

Normally the Translator service retains profanity that is present in the source in the translation. The degree of profanity and the context that makes words profane differ between cultures. As a result, the degree of profanity in the target language may be amplified or reduced.

If you want to avoid seeing profanity in the translation, even if profanity is present in the source text, use the profanity filtering option available in the `Translate()` method. This option allows you to choose whether you want to see profanity deleted, marked with appropriate tags, or take no action taken.

The `Translate()` method takes the "options" parameter, which contains the new element "ProfanityAction". The accepted values of ProfanityAction are "NoAction", "Marked" and "Deleted."

Accepted values of ProfanityAction and examples

PROFANIYACTION VALUE	ACTION	EXAMPLE: SOURCE - JAPANESE	EXAMPLE: TARGET - ENGLISH
NoAction	Default. Same as not setting the option. Profanity passes from source to target.	彼は変態です。	He is a jerk.
Marked	Profane words are surrounded by XML tags <profanity> ... </profanity>.	彼は変態です。	He is a <profanity>jerk</profanity> .
Deleted	Profane words are removed from the output without replacement.	彼は。	He is a.

Next steps

[Apply profanity filtering with your Translator API call](#)

How to receive word alignment information

11/8/2019 • 2 minutes to read • [Edit Online](#)

Receiving word alignment information

To receive alignment information, use the Translate method and include the optional includeAlignment parameter.

Alignment information format

Alignment is returned as a string value of the following format for every word of the source. The information for each word is separated by a space, including for non-space-separated languages (scripts) like Chinese:

`[[SourceTextStartIndex]:[SourceTextEndIndex]-[TgtTextStartIndex]:[TgtTextEndIndex]] *`

Example alignment string: "0:0-7:10 1:2-11:20 3:4-0:3 3:4-4:6 5:5-21:21".

In other words, the colon separates start and end index, the dash separates the languages, and space separates the words. One word may align with zero, one, or multiple words in the other language, and the aligned words may be non-contiguous. When no alignment information is available, the Alignment element will be empty. The method returns no error in that case.

Restrictions

Alignment is only returned for a subset of the language pairs at this point:

- from English to any other language;
- from any other language to English except for Chinese Simplified, Chinese Traditional, and Latvian to English
- from Japanese to Korean or from Korean to Japanese You will not receive alignment information if the sentence is a canned translation. Example of a canned translation is "This is a test", "I love you", and other high frequency sentences.

Example

Example JSON

```
[
  {
    "translations": [
      {
        "text": "Kann ich morgen Ihr Auto fahren?",
        "to": "de",
        "alignment": {
          "proj": "0:2-0:3 4:4-5:7 6:10-25:30 12:15-16:18 17:19-20:23 21:28-9:14 29:29-31:31"
        }
      }
    ]
  }
]
```

How to prevent translation of content with the Translator Text API

11/21/2019 • 2 minutes to read • [Edit Online](#)

The Translator Text API allows you to tag content so that it isn't translated. For example, you may want to tag code, a brand name, or a word/phrase that doesn't make sense when localized.

Methods for preventing translation

1. Escape to a Twitter tag `@somethingtopassthrough` or `#somethingtopassthrough`. Un-escape after translation. This is the regular expression for valid twitter tags: `\B@[A-Za-z]+[A-Za-z0-9_]+`. A tag should start with a "@" sign, followed by a character and then followed by one or many characters, digits or underscore. It is recommended to keep tags short and the opening tag must be preceded by a space.
2. Tag your content with `notranslate`. It's by design that this works only when the input `textType` is set as HTML

Example:

```
<span class="notranslate">This will not be translated.</span>
<span>This will be translated. </span>
```

```
<div class="notranslate">This will not be translated.</div>
<div>This will be translated. </div>
```

3. Use the [dynamic dictionary](#) to prescribe a specific translation.
4. Don't pass the string to the Translator Text API for translation.
5. Custom Translator: Use a [dictionary in Custom Translator](#) to prescribe the translation of a phrase with 100% probability.

Next steps

[Avoid translation in your Translator API call](#)

How to use a dynamic dictionary

12/10/2019 • 2 minutes to read • [Edit Online](#)

If you already know the translation you want to apply to a word or a phrase, you can supply it as markup within the request. The dynamic dictionary is safe only for compound nouns like proper names and product names.

Syntax:

```
<mstrans:dictionary translation="translation of phrase">phrase</mstrans:dictionary>
```

Requirements:

- The `From` and `To` languages must include English and another supported language.
- You must include the `From` parameter in your API translation request instead of using the autodetect feature.

Example: en-de:

Source input:

```
The word <mstrans:dictionary translation=\"wordomatic\">word or phrase</mstrans:dictionary> is a dictionary entry.
```

Target output: `Das Wort "wordomatic" ist ein Wörterbucheintrag.`

This feature works the same way with and without HTML mode.

Use the feature sparingly. A better way to customize translation is by using Custom Translator. Custom Translator makes full use of context and statistical probabilities. If you have or can create training data that shows your work or phrase in context, you get much better results. You can find more information about Custom Translator at <https://aka.ms/CustomTranslator>.

How to translate behind IP firewalls with the Translator Text API

11/8/2019 • 2 minutes to read • [Edit Online](#)

Translator Text API can translate behind firewalls using either domain-name or IP filtering. Domain-name filtering is the preferred method. We **do not recommend** running Microsoft Translator from behind an IP filtered firewall. The setup is likely to break in the future without notice.

Translator IP Addresses

The IP addresses for `api.cognitive.microsofttranslator.com` - Microsoft Translator Text API as of August 21, 2019:

- **Asia Pacific:** 20.40.125.208, 20.43.88.240, 20.184.58.62, 40.90.139.163, 104.44.89.44
- **Europe:** 40.90.138.4, 40.90.141.99, 51.105.170.64, 52.155.218.251
- **North America:** 40.90.139.36, 40.90.139.2, 40.119.2.134, 52.224.200.129, 52.249.207.163

Next steps

[Translate behind IP firewalls in your Translator API call](#)

Translator Text API v3.0

11/15/2019 • 7 minutes to read • [Edit Online](#)

What's new?

Version 3 of the Translator Text API provides a modern JSON-based Web API. It improves usability and performance by consolidating existing features into fewer operations and it provides new features.

- Transliteration to convert text in one language from one script to another script.
- Translation to multiple languages in one request.
- Language detection, translation, and transliteration in one request.
- Dictionary to look up alternative translations of a term, to find back-translations and examples showing terms used in context.
- More informative language detection results.

Base URLs

Microsoft Translator is served out of multiple datacenter locations. Currently they are located in 10 [Azure geographies](#):

- **Americas:** East US, South Central US, West Central US, and West US 2
- **Asia Pacific:** Korea South, Japan East, Southeast Asia, and Australia East
- **Europe:** North Europe and West Europe

Requests to the Microsoft Translator Text API are in most cases handled by the datacenter that is closest to where the request originated. In case of a datacenter failure, the request may be routed outside of the Azure geography.

To force the request to be handled by a specific Azure geography, change the Global endpoint in the API request to the desired regional endpoint:

DESCRIPTION	AZURE GEOGRAPHY	BASE URL
Azure	Global (non-regional)	api.cognitive.microsofttranslator.com
Azure	United States	api-nam.cognitive.microsofttranslator.com
Azure	Europe	api-eur.cognitive.microsofttranslator.com
Azure	Asia Pacific	api-apc.cognitive.microsofttranslator.com

Authentication

Subscribe to Translator Text API or [Cognitive Services multi-service](#) in Azure Cognitive Services, and use your subscription key (available in the Azure portal) to authenticate.

There are three headers that you can use to authenticate your subscription. This table describes how each is used:

HEADERS	DESCRIPTION
Ocp-Apim-Subscription-Key	Use with Cognitive Services subscription if you are passing your secret key. The value is the Azure secret key for your subscription to Translator Text API.
Authorization	Use with Cognitive Services subscription if you are passing an authentication token. The value is the Bearer token: <code>Bearer <token></code> .
Ocp-Apim-Subscription-Region	Use with Cognitive Services multi-service subscription if you are passing a multi-service secret key. The value is the region of the multi-service subscription. This value is optional when not using a multi-service subscription.

Secret key

The first option is to authenticate using the `Ocp-Apim-Subscription-Key` header. Add the `Ocp-Apim-Subscription-Key: <YOUR_SECRET_KEY>` header to your request.

Authorization token

Alternatively, you can exchange your secret key for an access token. This token is included with each request as the `Authorization` header. To obtain an authorization token, make a `POST` request to the following URL:

ENVIRONMENT	AUTHENTICATION SERVICE URL
Azure	<code>https://api.cognitive.microsoft.com/sts/v1.0/issueToken</code>

Here are example requests to obtain a token given a secret key:

```
// Pass secret key using header
curl --header 'Ocp-Apim-Subscription-Key: <your-key>' --data ""
'https://api.cognitive.microsoft.com/sts/v1.0/issueToken'

// Pass secret key using query string parameter
curl --data "" 'https://api.cognitive.microsoft.com/sts/v1.0/issueToken?Subscription-Key=<your-key>'
```

A successful request returns the encoded access token as plain text in the response body. The valid token is passed to the Translator service as a bearer token in the Authorization.

```
Authorization: Bearer <Base64-access_token>
```

An authentication token is valid for 10 minutes. The token should be reused when making multiple calls to the Translator APIs. However, if your program makes requests to the Translator API over an extended period of time, then your program must request a new access token at regular intervals (for example, every 8 minutes).

Multi-service subscription

The last authentication option is to use a Cognitive Service's multi-service subscription. This allows you to use a single secret key to authenticate requests for multiple services.

When you use a multi-service secret key, you must include two authentication headers with your request. The first passes the secret key, the second specifies the region associated with your subscription.

- `Ocp-Apim-Subscription-Key`
- `Ocp-Apim-Subscription-Region`

Region is required for the multi-service Text API subscription. The region you select is the only region that you can use for text translation when using the multi-service subscription key, and must be the same region you selected when you signed up for your multi-service subscription through the Azure portal.

Available regions are `australiaeast`, `brazilsouth`, `canadacentral`, `centralindia`, `centralus`, `centraluseuap`, `eastasia`, `eastus`, `eastus2`, `francecentral`, `japaneast`, `japanwest`, `koreacentral`, `northcentralus`, `northeurope`, `southcentralus`, `southeastasia`, `uksouth`, `westcentralus`, `westeurope`, `westus`, `westus2`, and `southafricanorth`.

If you pass the secret key in the query string with the parameter `Subscription-Key`, then you must specify the region with query parameter `Subscription-Region`.

If you use a bearer token, you must obtain the token from the region endpoint:

```
https://<your-region>.api.cognitive.microsoft.com/sts/v1.0/issueToken
```

Errors

A standard error response is a JSON object with name/value pair named `error`. The value is also a JSON object with properties:

- `code`: A server-defined error code.
- `message`: A string giving a human-readable representation of the error.

For example, a customer with a free trial subscription would receive the following error once the free quota is exhausted:

```
{
  "error": {
    "code": "403001",
    "message": "The operation is not allowed because the subscription has exceeded its free quota."
  }
}
```

The error code is a 6-digit number combining the 3-digit HTTP status code followed by a 3-digit number to further categorize the error. Common error codes are:

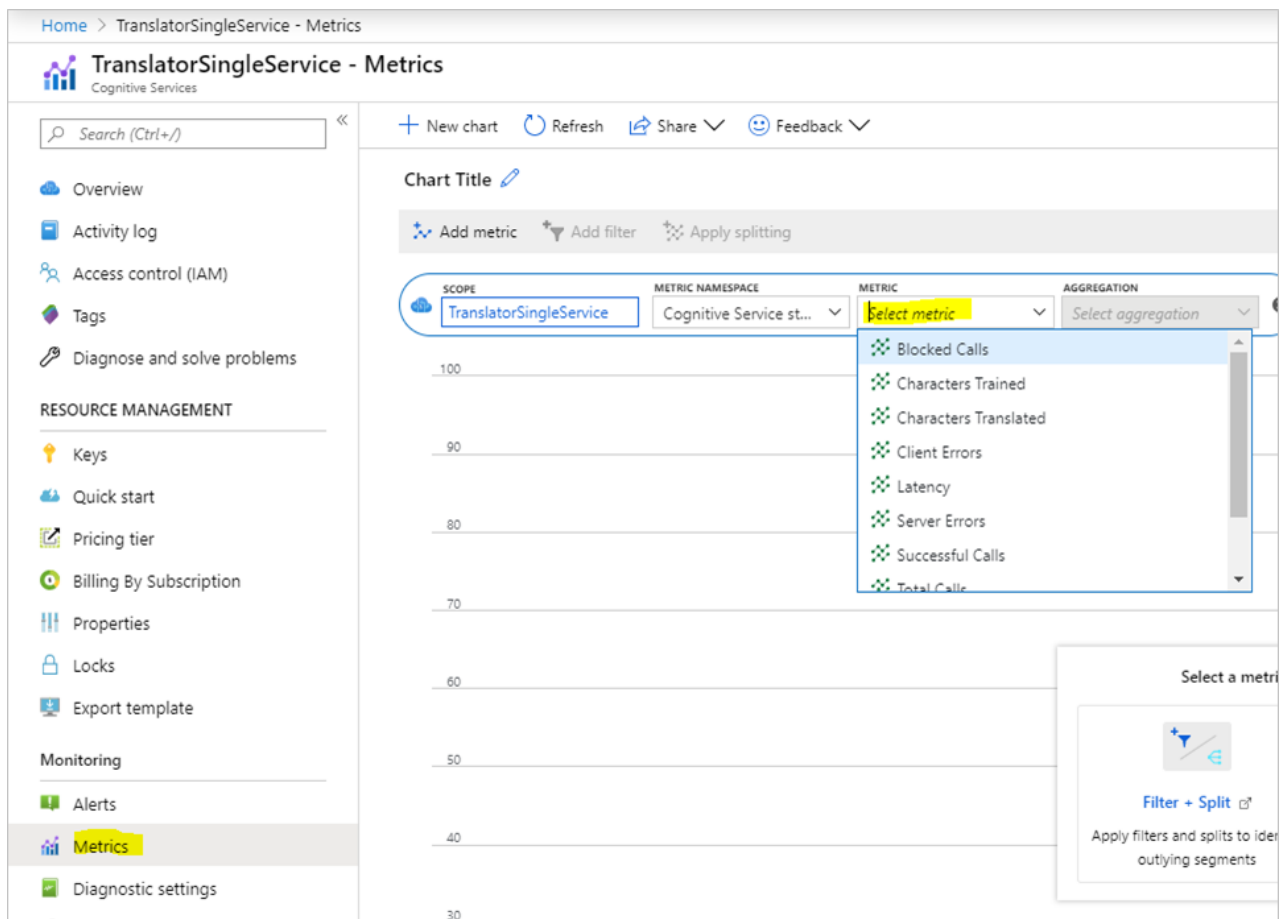
CODE	DESCRIPTION
400000	One of the request inputs is not valid.
400001	The "scope" parameter is invalid.
400002	The "category" parameter is invalid.
400003	A language specifier is missing or invalid.
400004	A target script specifier ("To script") is missing or invalid.
400005	An input text is missing or invalid.
400006	The combination of language and script is not valid.
400018	A source script specifier ("From script") is missing or invalid.
400019	One of the specified languages is not supported.

CODE	DESCRIPTION
400020	One of the elements in the array of input text is not valid.
400021	The API version parameter is missing or invalid.
400023	One of the specified language pair is not valid.
400035	The source language ("From" field) is not valid.
400036	The target language ("To" field) is missing or invalid.
400042	One of the options specified ("Options" field) is not valid.
400043	The client trace ID (ClientTraceId field or X-ClientTraceId header) is missing or invalid.
400050	The input text is too long. View request limits .
400064	The "translation" parameter is missing or invalid.
400070	The number of target scripts (ToScript parameter) does not match the number of target languages (To parameter).
400071	The value is not valid for TextType.
400072	The array of input text has too many elements.
400073	The script parameter is not valid.
400074	The body of the request is not valid JSON.
400075	The language pair and category combination is not valid.
400077	The maximum request size has been exceeded. View request limits .
400079	The custom system requested for translation between from and to language does not exist.
400080	Transliteration is not supported for the language or script.
401000	The request is not authorized because credentials are missing or invalid.
401015	"The credentials provided are for the Speech API. This request requires credentials for the Text API. Use a subscription to Translator Text API."
403000	The operation is not allowed.
403001	The operation is not allowed because the subscription has exceeded its free quota.

CODE	DESCRIPTION
405000	The request method is not supported for the requested resource.
408001	The translation system requested is being prepared. Please retry in a few minutes.
408002	Request timed out waiting on incoming stream. The client did not produce a request within the time that the server was prepared to wait. The client may repeat the request without modifications at any later time.
415000	The Content-Type header is missing or invalid.
429000, 429001, 429002	The server rejected the request because the client has exceeded request limits.
500000	An unexpected error occurred. If the error persists, report it with date/time of error, request identifier from response header X-RequestId, and client identifier from request header X-ClientTraceId.
503000	Service is temporarily unavailable. Please retry. If the error persists, report it with date/time of error, request identifier from response header X-RequestId, and client identifier from request header X-ClientTraceId.

Metrics

Metrics allow you to view the translator usage and availability information in Azure portal, under metrics section as shown in the below screenshot. For more information, see [Data and platform metrics](#).



This table lists available metrics with description of how they are used to monitor translation API calls.

METRICS	DESCRIPTION
TotalCalls	Total number of API calls.
TotalTokenCalls	Total number of API calls via token service using authentication token.
SuccessfulCalls	Number of successful calls.
TotalErrors	Number of calls with error response.
BlockedCalls	Number of calls that exceeded rate or quota limit.
ServerErrors	Number of calls with server internal error(5XX).
ClientErrors	Number of calls with client side error(4XX).
Latency	Duration to complete request in milliseconds.
CharactersTranslated	Total number of characters in incoming text request.

Translator Text API 3.0: Languages

11/8/2019 • 6 minutes to read • [Edit Online](#)

Gets the set of languages currently supported by other operations of the Translator Text API.

Request URL

Send a `GET` request to:

```
https://api.cognitive.microsofttranslator.com/languages?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

QUERY PARAMETER	DESCRIPTION
api-version	<i>Required parameter.</i> Version of the API requested by the client. Value must be `3.0`.
scope	<i>*Optional parameter*.</i> A comma-separated list of names defining the group of languages to return. Allowed group names are: `translation`, `transliteration` and `dictionary`. If no scope is given, then all groups are returned, which is equivalent to passing `scope=translation,transliteration,dictionary`. To decide which set of supported languages is appropriate for your scenario, see the description of the [response object] (#response-body).

Request headers are:

HEADERS	DESCRIPTION
Accept-Language	<i>*Optional request header*.</i> The language to use for user interface strings. Some of the fields in the response are names of languages or names of regions. Use this parameter to define the language in which these names are returned. The language is specified by providing a well-formed BCP 47 language tag. For instance, use the value `fr` to request names in French or use the value `zh-Hant` to request names in Chinese Traditional. Names are provided in the English language when a target language is not specified or when localization is not available.
X-ClientTraceId	<i>*Optional request header*.</i> A client-generated GUID to uniquely identify the request.

Authentication isn't required to get language resources.

Response body

A client uses the `scope` query parameter to define which groups of languages it's interested in.

- `scope=translation` provides languages supported to translate text from one language to another language;
- `scope=transliteration` provides capabilities for converting text in one language from one script to another script;

- `scope=dictionary` provides language pairs for which `Dictionary` operations return data.

A client may retrieve several groups simultaneously by specifying a comma-separated list of names. For example, `scope=translation,transliteration,dictionary` would return supported languages for all groups.

A successful response is a JSON object with one property for each requested group:

```
{
  "translation": {
    //... set of languages supported to translate text (scope=translation)
  },
  "transliteration": {
    //... set of languages supported to convert between scripts (scope=transliteration)
  },
  "dictionary": {
    //... set of languages supported for alternative translations and examples (scope=dictionary)
  }
}
```

The value for each property is as follows.

- `translation` property

The value of the `translation` property is a dictionary of (key, value) pairs. Each key is a BCP 47 language tag. A key identifies a language for which text can be translated to or translated from. The value associated with the key is a JSON object with properties that describe the language:

- `name`: Display name of the language in the locale requested via `Accept-Language` header.
- `nativeName`: Display name of the language in the locale native for this language.
- `dir`: Directionality, which is `rtl` for right-to-left languages or `ltr` for left-to-right languages.

An example is:

```
{
  "translation": {
    ...
    "fr": {
      "name": "French",
      "nativeName": "Français",
      "dir": "ltr"
    },
    ...
  }
}
```

- `transliteration` property

The value of the `transliteration` property is a dictionary of (key, value) pairs. Each key is a BCP 47 language tag. A key identifies a language for which text can be converted from one script to another script. The value associated with the key is a JSON object with properties that describe the language and its supported scripts:

- `name`: Display name of the language in the locale requested via `Accept-Language` header.
- `nativeName`: Display name of the language in the locale native for this language.
- `scripts`: List of scripts to convert from. Each element of the `scripts` list has properties:

- `code`: Code identifying the script.
- `name`: Display name of the script in the locale requested via `Accept-Language` header.
- `nativeName`: Display name of the language in the locale native for the language.
- `dir`: Directionality, which is `rtl` for right-to-left languages or `ltr` for left-to-right languages.
- `toScripts`: List of scripts available to convert text to. Each element of the `toScripts` list has properties `code`, `name`, `nativeName`, and `dir` as described earlier.

An example is:

```
{
  "transliteration": {
    ...
    "ja": {
      "name": "Japanese",
      "nativeName": "日本語",
      "scripts": [
        {
          "code": "Jpan",
          "name": "Japanese",
          "nativeName": "日本語",
          "dir": "ltr",
          "toScripts": [
            {
              "code": "Latn",
              "name": "Latin",
              "nativeName": "ラテン語",
              "dir": "ltr"
            }
          ]
        }
      ]
    },
    {
      "code": "Latn",
      "name": "Latin",
      "nativeName": "ラテン語",
      "dir": "ltr",
      "toScripts": [
        {
          "code": "Jpan",
          "name": "Japanese",
          "nativeName": "日本語",
          "dir": "ltr"
        }
      ]
    }
  ]
},
  ...
}
```

- `dictionary` property

The value of the `dictionary` property is a dictionary of (key, value) pairs. Each key is a BCP 47 language tag. The key identifies a language for which alternative translations and back-translations are available. The value is a JSON object that describes the source language and the target languages with available translations:

- `name`: Display name of the source language in the locale requested via `Accept-Language` header.

- `nativeName`: Display name of the language in the locale native for this language.
- `dir`: Directionality, which is `rtl` for right-to-left languages or `ltr` for left-to-right languages.
- `translations`: List of languages with alternative translations and examples for the query expressed in the source language. Each element of the `translations` list has properties:
 - `name`: Display name of the target language in the locale requested via `Accept-Language` header.
 - `nativeName`: Display name of the target language in the locale native for the target language.
 - `dir`: Directionality, which is `rtl` for right-to-left languages or `ltr` for left-to-right languages.
 - `code`: Language code identifying the target language.

An example is:

```
"es": {
  "name": "Spanish",
  "nativeName": "Español",
  "dir": "ltr",
  "translations": [
    {
      "name": "English",
      "nativeName": "English",
      "dir": "ltr",
      "code": "en"
    }
  ]
},
```

The structure of the response object will not change without a change in the version of the API. For the same version of the API, the list of available languages may change over time because Microsoft Translator continually extends the list of languages supported by its services.

The list of supported languages will not change frequently. To save network bandwidth and improve responsiveness, a client application should consider caching language resources and the corresponding entity tag (`ETag`). Then, the client application can periodically (for example, once every 24 hours) query the service to fetch the latest set of supported languages. Passing the current `ETag` value in an `If-None-Match` header field will allow the service to optimize the response. If the resource has not been modified, the service will return status code 304 and an empty response body.

Response headers

HEADERS	DESCRIPTION
<code>ETag</code>	Current value of the entity tag for the requested groups of supported languages. To make subsequent requests more efficient, the client may send the <code>ETag</code> value in an <code>If-None-Match</code> header field.
<code>X-RequestId</code>	Value generated by the service to identify the request. It is used for troubleshooting purposes.

Response status codes

The following are the possible HTTP status codes that a request returns.

STATUS CODE	DESCRIPTION
200	Success.
304	The resource has not been modified since the version specified by request headers `If-None-Match`.
400	One of the query parameters is missing or not valid. Correct request parameters before retrying.
429	The server rejected the request because the client has exceeded request limits.
500	An unexpected error occurred. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.
503	Server temporarily unavailable. Retry the request. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.

If an error occurs, the request will also return a JSON error response. The error code is a 6-digit number combining the 3-digit HTTP status code followed by a 3-digit number to further categorize the error. Common error codes can be found on the [v3 Translator Text API reference page](#).

Examples

The following example shows how to retrieve languages supported for text translation.

```
curl "https://api.cognitive.microsofttranslator.com/languages?api-version=3.0&scope=translation"
```

Translator Text API 3.0: Translate

11/19/2019 • 13 minutes to read • [Edit Online](#)

Translates text.

Request URL

Send a `POST` request to:

```
https://api.cognitive.microsofttranslator.com/translate?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

Required parameters

QUERY PARAMETER	DESCRIPTION
api-version	<i>Required parameter.</i> Version of the API requested by the client. Value must be <code>3.0</code> .
to	<i>Required parameter.</i> Specifies the language of the output text. The target language must be one of the supported languages included in the <code>translation</code> scope. For example, use <code>to=de</code> to translate to German. It's possible to translate to multiple languages simultaneously by repeating the parameter in the query string. For example, use <code>to=de&to=it</code> to translate to German and Italian.

Optional parameters

QUERY PARAMETER	DESCRIPTION
from	<i>Optional parameter.</i> Specifies the language of the input text. Find which languages are available to translate from by looking up supported languages using the <code>translation</code> scope. If the <code>from</code> parameter is not specified, automatic language detection is applied to determine the source language. You must use the <code>from</code> parameter rather than autodetection when using the dynamic dictionary feature.
textType	<i>Optional parameter.</i> Defines whether the text being translated is plain text or HTML text. Any HTML needs to be a well-formed, complete element. Possible values are: <code>plain</code> (default) or <code>html</code> .
category	<i>Optional parameter.</i> A string specifying the category (domain) of the translation. This parameter is used to get translations from a customized system built with Custom Translator . Add the Category ID from your Custom Translator project details to this parameter to use your deployed customized system. Default value is: <code>general</code> .

profanityAction	<p><i>Optional parameter.</i></p> <p>Specifies how profanities should be treated in translations. Possible values are: <code>NoAction</code> (default), <code>Marked</code> or <code>Deleted</code>. To understand ways to treat profanity, see Profanity handling.</p>
profanityMarker	<p><i>Optional parameter.</i></p> <p>Specifies how profanities should be marked in translations. Possible values are: <code>Asterisk</code> (default) or <code>Tag</code>. To understand ways to treat profanity, see Profanity handling.</p>
includeAlignment	<p><i>Optional parameter.</i></p> <p>Specifies whether to include alignment projection from source text to translated text. Possible values are: <code>true</code> or <code>false</code> (default).</p>
includeSentenceLength	<p><i>Optional parameter.</i></p> <p>Specifies whether to include sentence boundaries for the input text and the translated text. Possible values are: <code>true</code> or <code>false</code> (default).</p>
suggestedFrom	<p><i>Optional parameter.</i></p> <p>Specifies a fallback language if the language of the input text can't be identified. Language auto-detection is applied when the <code>from</code> parameter is omitted. If detection fails, the <code>suggestedFrom</code> language will be assumed.</p>
fromScript	<p><i>Optional parameter.</i></p> <p>Specifies the script of the input text.</p>
toScript	<p><i>Optional parameter.</i></p> <p>Specifies the script of the translated text.</p>
allowFallback	<p><i>Optional parameter.</i></p> <p>Specifies that the service is allowed to fallback to a general system when a custom system does not exist. Possible values are: <code>true</code> (default) or <code>false</code>.</p> <p><code>allowFallback=false</code> specifies that the translation should only use systems trained for the <code>category</code> specified by the request. If a translation for language X to language Y requires chaining through a pivot language E, then all the systems in the chain (X->E and E->Y) will need to be custom and have the same category. If no system is found with the specific category, the request will return a 400 status code. <code>allowFallback=true</code> specifies that the service is allowed to fallback to a general system when a custom system does not exist.</p>

Request headers include:

HEADERS	DESCRIPTION
Authentication header(s)	<p><i>Required request header.</i></p> <p>See available options for authentication.</p>
Content-Type	<p><i>Required request header.</i></p> <p>Specifies the content type of the payload.</p> <p>Accepted value is <code>application/json; charset=UTF-8</code>.</p>
Content-Length	<p><i>Required request header.</i></p> <p>The length of the request body.</p>
X-ClientTraceId	<p><i>Optional.</i></p> <p>A client-generated GUID to uniquely identify the request. You can omit this header if you include the trace ID in the query string using a query parameter named <code>ClientTraceId</code>.</p>

Request body

The body of the request is a JSON array. Each array element is a JSON object with a string property named `Text`, which represents the string to translate.

```
[
  {
    "Text": "I would really like to drive your car around the block a few times."
  }
]
```

The following limitations apply:

- The array can have at most 100 elements.
- The entire text included in the request cannot exceed 5,000 characters including spaces.

Response body

A successful response is a JSON array with one result for each string in the input array. A result object includes the following properties:

- `detectedLanguage`: An object describing the detected language through the following properties:
 - `language`: A string representing the code of the detected language.
 - `score`: A float value indicating the confidence in the result. The score is between zero and one and a low score indicates a low confidence.

The `detectedLanguage` property is only present in the result object when language auto-detection is requested.

- `translations`: An array of translation results. The size of the array matches the number of target languages specified through the `to` query parameter. Each element in the array includes:
 - `to`: A string representing the language code of the target language.
 - `text`: A string giving the translated text.
 - `transliteration`: An object giving the translated text in the script specified by the `toScript` parameter.
 - `script`: A string specifying the target script.
 - `text`: A string giving the translated text in the target script.

The `transliteration` object is not included if transliteration does not take place.

- `alignment`: An object with a single string property named `proj`, which maps input text to translated text. The alignment information is only provided when the request parameter `includeAlignment` is `true`. Alignment is returned as a string value of the following format: `[[SourceTextStartIndex]:[SourceTextEndIndex]-[TgtTextStartIndex]:[TgtTextEndIndex]]`. The colon separates start and end index, the dash separates the languages, and space separates the words. One word may align with zero, one, or multiple words in the other language, and the aligned words may be non-contiguous. When no alignment information is available, the alignment element will be empty. See [Obtain alignment information](#) for an example and restrictions.
- `sentLen`: An object returning sentence boundaries in the input and output texts.
 - `srcSentLen`: An integer array representing the lengths of the sentences in the input text. The length of the array is the number of sentences, and the values are the length of each

sentence.

- `transSentLen`: An integer array representing the lengths of the sentences in the translated text. The length of the array is the number of sentences, and the values are the length of each sentence.

Sentence boundaries are only included when the request parameter `includeSentenceLength` is `true`.

- `sourceText`: An object with a single string property named `text`, which gives the input text in the default script of the source language. `sourceText` property is present only when the input is expressed in a script that's not the usual script for the language. For example, if the input were Arabic written in Latin script, then `sourceText.text` would be the same Arabic text converted into Arab script.

Example of JSON responses are provided in the [examples](#) section.

Response headers

HEADERS	DESCRIPTION
X-RequestId	Value generated by the service to identify the request. It is used for troubleshooting purposes.
X-MT-System	Specifies the system type that was used for translation for each 'to' language requested for translation. The value is a comma-separated list of strings. Each string indicates a type: <ul style="list-style-type: none">• Custom - Request includes a custom system and at least one custom system was used during translation.• Team - All other requests

Response status codes

The following are the possible HTTP status codes that a request returns.

STATUS CODE	DESCRIPTION
200	Success.
400	One of the query parameters is missing or not valid. Correct request parameters before retrying.
401	The request could not be authenticated. Check that credentials are specified and valid.
403	The request is not authorized. Check the details error message. This often indicates that all free translations provided with a trial subscription have been used up.
408	The request could not be fulfilled because a resource is missing. Check the details error message. When using a custom <code>category</code> , this often indicates that the custom translation system is not yet available to serve requests. The request should be retried after a waiting period (e.g. 1 minute).
429	The server rejected the request because the client has exceeded request limits.
500	An unexpected error occurred. If the error persists, report it with: date and time of the failure, request identifier from response header <code>X-RequestId</code> , and client identifier from request header <code>X-ClientTraceId</code> .
503	Server temporarily unavailable. Retry the request. If the error persists, report it with: date and time of the failure, request identifier from response header <code>X-RequestId</code> , and client identifier from request header <code>X-ClientTraceId</code> .

If an error occurs, the request will also return a JSON error response. The error code is a 6-digit number combining the 3-digit HTTP status code followed by a 3-digit number to further categorize the error. Common error codes can be found on the [v3 Translator Text API reference page](#).

Examples

Translate a single input

This example shows how to translate a single sentence from English to Simplified Chinese.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=zh-Hans" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hello, what is your name?'}]"
```

The response body is:

```
[
  {
    "translations": [
      { "text": "你好，你叫什么名字？", "to": "zh-Hans" }
    ]
  }
]
```

The `translations` array includes one element, which provides the translation of the single piece of text in the input.

Translate a single input with language auto-detection

This example shows how to translate a single sentence from English to Simplified Chinese. The request does not specify the input language. Auto-detection of the source language is used instead.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&to=zh-Hans" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hello, what is your name?'}]"
```

The response body is:

```
[
  {
    "detectedLanguage": { "language": "en", "score": 1.0 },
    "translations": [
      { "text": "你好，你叫什么名字？", "to": "zh-Hans" }
    ]
  }
]
```

The response is similar to the response from the previous example. Since language auto-detection was requested, the response also includes information about the language detected for the input text.

Translate with transliteration

Let's extend the previous example by adding transliteration. The following request asks for a Chinese translation written in Latin script.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&to=zh-Hans&toScript=Latn" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hello, what is your name?'}]"
```

The response body is:

```
[
  {
    "detectedLanguage":{"language":"en","score":1.0},
    "translations":[
      {
        "text":"你好，你叫什么名字？",
        "transliteration":{"script":"Latn", "text":"nǐ hǎo , nǐ jiào shén me míng zì ?"},
        "to":"zh-Hans"
      }
    ]
  }
]
```

The translation result now includes a `transliteration` property, which gives the translated text using Latin characters.

Translate multiple pieces of text

Translating multiple strings at once is simply a matter of specifying an array of strings in the request body.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=zh-Hans" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hello, what is your name?'}, {'Text':'I am fine, thank you.'}]"
```

The response body is:

```
[
  {
    "translations":[
      {"text":"你好，你叫什么名字？","to":"zh-Hans"}
    ]
  },
  {
    "translations":[
      {"text":"我很好，谢谢你。","to":"zh-Hans"}
    ]
  }
]
```

Translate to multiple languages

This example shows how to translate the same input to several languages in one request.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=zh-Hans&to=de" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hello, what is your name?'}]"
```

The response body is:


```
[
  {
    "translations":[
      {"text":"你好，你叫什么名字？","to":"zh-Hans"},
      {"text":"Hallo, was ist dein Name?","to":"de"}
    ]
  }
]
```

Handle profanity

Normally the Translator service will retain profanity that is present in the source in the translation. The degree of profanity and the context that makes words profane differ between cultures, and as a result the degree of profanity in the target language may be amplified or reduced.

If you want to avoid getting profanity in the translation, regardless of the presence of profanity in the source text, you can use the profanity filtering option. The option allows you to choose whether you want to see profanity deleted, whether you want to mark profanities with appropriate tags (giving you the option to add your own post-processing), or you want no action taken. The accepted values of `ProfanityAction` are `Deleted`, `Marked` and `NoAction` (default).

PROFANITYACTION	ACTION
<code>NoAction</code>	<p>This is the default behavior. Profanity will pass from source to target.</p> <p>Example Source (Japanese): 彼はジャッカスです。 Example Translation (English): He is a jackass.</p>
<code>Deleted</code>	<p>Profane words will be removed from the output without replacement.</p> <p>Example Source (Japanese): 彼はジャッカスです。 Example Translation (English): He is a.</p>
<code>Marked</code>	<p>Profane words are replaced by a marker in the output. The marker depends on the <code>ProfanityMarker</code> parameter.</p> <p>For <code>ProfanityMarker=Asterisk</code>, profane words are replaced with <code>***</code>:</p> <p>Example Source (Japanese): 彼はジャッカスです。 Example Translation (English): He is a ***.</p> <p>For <code>ProfanityMarker=Tag</code>, profane words are surrounded by XML tags <code><profanity></code> and <code></profanity></code>:</p> <p>Example Source (Japanese): 彼はジャッカスです。 Example Translation (English): He is a <profanity>jackass</profanity>.</p>

For example:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=de&profanityAction=Marked" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'This is a freaking good idea.'}]"
```

This returns:

```
[
  {
    "translations":[
      {"text":"Das ist eine *** gute Idee.", "to":"de"}
    ]
  }
]
```

Compare with:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=de&profanityAction=Marked&profanityMarker=Tag" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'This is a freaking good idea.'}]"
```

That last request returns:

```
[
  {
    "translations":[
      {"text":"Das ist eine <profanity>verdammt</profanity> gute Idee.", "to":"de"}
    ]
  }
]
```

Translate content with markup and decide what's translated

It's common to translate content which includes markup such as content from an HTML page or content from an XML document. Include query parameter `textType=html` when translating content with tags. In addition, it's sometimes useful to exclude specific content from translation. You can use the attribute `class=nottranslate` to specify content that should remain in its original language. In the following example, the content inside the first `div` element will not be translated, while the content in the second `div` element will be translated.

```
<div class="nottranslate">This will not be translated.</div>
<div>This will be translated. </div>
```

Here is a sample request to illustrate.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=zh-Hans&textType=html" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'<div class=\\"nottranslate\\">This will not be translated.</div><div>This will be translated.</div>'}]"
```

The response is:

```
[
  {
    "translations":[
      {"text":"<div class=\\"nottranslate\\">This will not be translated.</div><div>这将被翻译。</div>", "to":"zh-Hans"}
    ]
  }
]
```

Obtain alignment information

To receive alignment information, specify `includeAlignment=true` on the query string.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=fr&includeAlignment=true" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'The answer lies in machine translation.'}]"
```

The response is:

```
[
  {
    "translations": [
      {
        "text": "La réponse se trouve dans la traduction automatique.",
        "to": "fr",
        "alignment": {"proj": "0:2-0:1 4:9-3:9 11:14-11:19 16:17-21:24 19:25-40:50 27:37-29:38 38:38-51:51"}
      }
    ]
  }
]
```

The alignment information starts with `0:2-0:1`, which means that the first three characters in the source text (`The`) map to the first two characters in the translated text (`La`).

Limitations

Note the following restrictions:

- Alignment is not available for text in HTML format i.e., `textType=html`
- Alignment is only returned for a subset of the language pairs:
 - from English to any other language;
 - from any other language to English except for Chinese Simplified, Chinese Traditional, and Latvian to English;
 - from Japanese to Korean or from Korean to Japanese.
- You will not receive alignment if the sentence is a canned translation. Example of a canned translation is "This is a test", "I love you" and other high frequency sentences.

Obtain sentence boundaries

To receive information about sentence length in the source text and translated text, specify

`includeSentenceLength=true` on the query string.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=fr&includeSentenceLength=true" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'The answer lies in machine translation. The best machine translation technology cannot always provide translations tailored to a site or users like a human. Simply copy and paste a code snippet anywhere.'}]"
```

The response is:

```
[
  {
    "translations":[
      {
        "text":"La réponse se trouve dans la traduction automatique. La meilleure technologie de
traduction automatique ne peut pas toujours fournir des traductions adaptées à un site ou des utilisateurs
comme un être humain. Il suffit de copier et coller un extrait de code n'importe où.",
        "to":"fr",
        "sentLen":{"srcSentLen":[40,117,46],"transSentLen":[53,157,62]}
      }
    ]
  }
]
```

Translate with dynamic dictionary

If you already know the translation you want to apply to a word or a phrase, you can supply it as markup within the request. The dynamic dictionary is only safe for compound nouns like proper names and product names.

The markup to supply uses the following syntax.

```
<mstrans:dictionary translation="translation of phrase">phrase</mstrans:dictionary>
```

For example, consider the English sentence "The word wordomatic is a dictionary entry." To preserve the word *wordomatic* in the translation, send the request:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=de" -H "Ocp-
Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'The
word <mstrans:dictionary translation=\"wordomatic\">word or phrase</mstrans:dictionary> is a dictionary
entry.'}]"
```

The result is:

```
[
  {
    "translations":[
      {"text":"Das Wort \"wordomatic\" ist ein Wörterbucheintrag.", "to":"de"}
    ]
  }
]
```

This feature works the same way with `textType=text` or with `textType=html`. The feature should be used sparingly. The appropriate and far better way of customizing translation is by using Custom Translator. Custom Translator makes full use of context and statistical probabilities. If you have or can afford to create training data that shows your work or phrase in context, you get much better results. [Learn more about Custom Translator.](#)

Translator Text API 3.0: Transliterate

11/8/2019 • 3 minutes to read • [Edit Online](#)

Converts text in one language from one script to another script.

Request URL

Send a `POST` request to:

```
https://api.cognitive.microsofttranslator.com/transliterate?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

QUERY PARAMETER	DESCRIPTION
api-version	*Required parameter*. Version of the API requested by the client. Value must be `3.0`.
language	*Required parameter*. Specifies the language of the text to convert from one script to another. Possible languages are listed in the `transliteration` scope obtained by querying the service for its [supported languages](./v3-0-languages.md).
fromScript	*Required parameter*. Specifies the script used by the input text. Look up [supported languages](./v3-0-languages.md) using the `transliteration` scope, to find input scripts available for the selected language.
toScript	*Required parameter*. Specifies the output script. Look up [supported languages](./v3-0-languages.md) using the `transliteration` scope, to find output scripts available for the selected combination of input language and input script.

Request headers include:

HEADERS	DESCRIPTION
Authentication header(s)	<i>Required request header.</i> See available options for authentication .
Content-Type	*Required request header*. Specifies the content type of the payload. Possible values are: `application/json`.
Content-Length	*Required request header*. The length of the request body.
X-ClientTraceId	*Optional*. A client-generated GUID to uniquely identify the request. Note that you can omit this header if you include the trace ID in the query string using a query parameter named `ClientTraceId`.

Request body

The body of the request is a JSON array. Each array element is a JSON object with a string property named `Text`, which represents the string to convert.

```
[
  {
    "Text": "こんにちは",
  },
  {
    "Text": "さようなら"
  }
]
```

The following limitations apply:

- The array can have at most 10 elements.
- The text value of an array element cannot exceed 1,000 characters including spaces.
- The entire text included in the request cannot exceed 5,000 characters including spaces.

Response body

A successful response is a JSON array with one result for each element in the input array. A result object includes the following properties:

- `text`: A string which is the result of converting the input string to the output script.
- `script`: A string specifying the script used in the output.

An example JSON response is:

```
[
  {
    "text": "konnichiha",
    "script": "Latn"
  },
  {
    "text": "sayounara",
    "script": "Latn"
  }
]
```

Response headers

HEADERS	DESCRIPTION
X-RequestId	Value generated by the service to identify the request. It is used for troubleshooting purposes.

Response status codes

The following are the possible HTTP status codes that a request returns.

STATUS CODE	DESCRIPTION
200	Success.
400	One of the query parameters is missing or not valid. Correct request parameters before retrying.
401	The request could not be authenticated. Check that credentials are specified and valid.
403	The request is not authorized. Check the details error message. This often indicates that all free translations provided with a trial subscription have been used up.
429	The server rejected the request because the client has exceeded request limits.

500	An unexpected error occurred. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.
503	Server temporarily unavailable. Retry the request. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.

If an error occurs, the request will also return a JSON error response. The error code is a 6-digit number combining the 3-digit HTTP status code followed by a 3-digit number to further categorize the error. Common error codes can be found on the [v3 Translator Text API reference page](#).

Examples

The following example shows how to convert two Japanese strings into Romanized Japanese.

The JSON payload for the request in this example:

```
[{"text":"こんにちは","script":"jpan"}, {"text":"さようなら","script":"jpan"}]
```

If you are using cURL in a command-line window that does not support Unicode characters, take the following JSON payload and save it into a file named `request.txt`. Be sure to save the file with `UTF-8` encoding.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/transliterate?api-version=3.0&language=ja&fromScript=Jpan&toScript=Latn" -H "X-ClientTraceId: 875030C7-5380-40B8-8A03-63DACC69C11" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json" -d @request.txt
```

Translator Text API 3.0: Detect

11/8/2019 • 3 minutes to read • [Edit Online](#)

Identifies the language of a piece of text.

Request URL

Send a `POST` request to:

```
https://api.cognitive.microsofttranslator.com/detect?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

QUERY PARAMETER	DESCRIPTION
api-version	*Required parameter*. Version of the API requested by the client. Value must be `3.0`.

Request headers include:

HEADERS	DESCRIPTION
Authentication header(s)	<i>Required request header.</i> See available options for authentication .
Content-Type	*Required request header*. Specifies the content type of the payload. Possible values are: `application/json`.
Content-Length	*Required request header*. The length of the request body.
X-ClientTraceId	*Optional*. A client-generated GUID to uniquely identify the request. Note that you can omit this header if you include the trace ID in the query string using a query parameter named `ClientTraceId`.

Request body

The body of the request is a JSON array. Each array element is a JSON object with a string property named `Text`. Language detection is applied to the value of the `Text` property. A sample request body looks like that:

```
[
  { "Text": "Ich würde wirklich gern Ihr Auto um den Block fahren ein paar Mal." }
]
```

The following limitations apply:

- The array can have at most 100 elements.
- The text value of an array element cannot exceed 10,000 characters including spaces.

- The entire text included in the request cannot exceed 50,000 characters including spaces.

Response body

A successful response is a JSON array with one result for each string in the input array. A result object includes the following properties:

- `language`: Code of the detected language.
- `score`: A float value indicating the confidence in the result. The score is between zero and one and a low score indicates a low confidence.
- `isTranslationSupported`: A boolean value which is true if the detected language is one of the languages supported for text translation.
- `isTransliterationSupported`: A boolean value which is true if the detected language is one of the languages supported for transliteration.
- `alternatives`: An array of other possible languages. Each element of the array is another object with the same properties listed above: `language`, `score`, `isTranslationSupported` and `isTransliterationSupported`.

An example JSON response is:

```
[
  {
    "language": "de",
    "score": 0.92,
    "isTranslationSupported": true,
    "isTransliterationSupported": false,
    "alternatives": [
      {
        "language": "pt",
        "score": 0.23,
        "isTranslationSupported": true,
        "isTransliterationSupported": false
      },
      {
        "language": "sk",
        "score": 0.23,
        "isTranslationSupported": true,
        "isTransliterationSupported": false
      }
    ]
  }
]
```

Response headers

HEADERS	DESCRIPTION
X-RequestId	Value generated by the service to identify the request. It is used for troubleshooting purposes.

Response status codes

The following are the possible HTTP status codes that a request returns.

STATUS CODE	DESCRIPTION
-------------	-------------

200	Success.
400	One of the query parameters is missing or not valid. Correct request parameters before retrying.
401	The request could not be authenticated. Check that credentials are specified and valid.
403	The request is not authorized. Check the details error message. This often indicates that all free translations provided with a trial subscription have been used up.
429	The server rejected the request because the client has exceeded request limits.
500	An unexpected error occurred. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.
503	Server temporarily unavailable. Retry the request. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.

If an error occurs, the request will also return a JSON error response. The error code is a 6-digit number combining the 3-digit HTTP status code followed by a 3-digit number to further categorize the error. Common error codes can be found on the [v3 Translator Text API reference page](#).

Examples

The following example shows how to retrieve languages supported for text translation.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/detect?api-version=3.0" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json" -d "[{'Text':'What language is this text written in?'}]"
```

Translator Text API 3.0: BreakSentence

12/10/2019 • 3 minutes to read • [Edit Online](#)

Identifies the positioning of sentence boundaries in a piece of text.

Request URL

Send a `POST` request to:

```
https://api.cognitive.microsofttranslator.com/breaksentence?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

QUERY PARAMETER	DESCRIPTION
api-version	*Required query parameter*. Version of the API requested by the client. Value must be `3.0`.
language	*Optional query parameter*. Language tag identifying the language of the input text. If a code is not specified, automatic language detection will be applied.
script	*Optional query parameter*. Script tag identifying the script used by the input text. If a script is not specified, the default script of the language will be assumed.

Request headers include:

HEADERS	DESCRIPTION
Authentication header(s)	<i>Required request header.</i> See available options for authentication .
Content-Type	*Required request header*. Specifies the content type of the payload. Possible values are: `application/json`.
Content-Length	*Required request header*. The length of the request body.
X-ClientTraceId	*Optional*. A client-generated GUID to uniquely identify the request. Note that you can omit this header if you include the trace ID in the query string using a query parameter named `ClientTraceId`.

Request body

The body of the request is a JSON array. Each array element is a JSON object with a string property named `Text`. Sentence boundaries are computed for the value of the `Text` property. A sample request body with one piece of text looks like that:

```
[
  { "Text": "How are you? I am fine. What did you do today?" }
]
```

The following limitations apply:

- The array can have at most 100 elements.
- The text value of an array element cannot exceed 10,000 characters including spaces.
- The entire text included in the request cannot exceed 50,000 characters including spaces.
- If the `language` query parameter is specified, then all array elements must be in the same language. Otherwise, language auto-detection is applied to each array element independently.

Response body

A successful response is a JSON array with one result for each string in the input array. A result object includes the following properties:

- `sentLen`: An array of integers representing the lengths of the sentences in the text element. The length of the array is the number of sentences, and the values are the length of each sentence.
- `detectedLanguage`: An object describing the detected language through the following properties:
 - `language`: Code of the detected language.
 - `score`: A float value indicating the confidence in the result. The score is between zero and one and a low score indicates a low confidence.

Note that the `detectedLanguage` property is only present in the result object when language auto-detection is requested.

An example JSON response is:

```
[
  {
    "sentLen": [ 13, 11, 22 ]
    "detectedLanguage": {
      "language": "en",
      "score": 401
    },
  }
]
```

Response headers

HEADERS	DESCRIPTION
X-RequestId	Value generated by the service to identify the request. It is used for troubleshooting purposes.

Response status codes

The following are the possible HTTP status codes that a request returns.

STATUS CODE	DESCRIPTION
200	Success.

400	One of the query parameters is missing or not valid. Correct request parameters before retrying.
401	The request could not be authenticated. Check that credentials are specified and valid.
403	The request is not authorized. Check the details error message. This often indicates that all free translations provided with a trial subscription have been used up.
429	The server rejected the request because the client has exceeded request limits.
500	An unexpected error occurred. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.
503	Server temporarily unavailable. Retry the request. If the error persists, report it with: date and time of the failure, request identifier from response header `X-RequestId`, and client identifier from request header `X-ClientTraceId`.

If an error occurs, the request will also return a JSON error response. The error code is a 6-digit number combining the 3-digit HTTP status code followed by a 3-digit number to further categorize the error. Common error codes can be found on the [v3 Translator Text API reference page](#).

Examples

The following example shows how to obtain sentence boundaries for a single sentence. The language of the sentence is automatically detected by the service.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/breaksentence?api-version=3.0" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json" -d "[{'Text': 'How are you? I am fine. What did you do today?'}]"
```

Translator Text API 3.0: Dictionary Lookup

11/8/2019 • 5 minutes to read • [Edit Online](#)

Provides alternative translations for a word and a small number of idiomatic phrases. Each translation has a part-of-speech and a list of back-translations. The back-translations enable a user to understand the translation in context. The [Dictionary Example](#) operation allows further drill down to see example uses of each translation pair.

Request URL

Send a `POST` request to:

```
https://api.cognitive.microsofttranslator.com/dictionary/lookup?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

QUERY PARAMETER	DESCRIPTION
api-version	*Required parameter*. Version of the API requested by the client. Value must be `3.0`.
from	*Required parameter*. Specifies the language of the input text. The source language must be one of the [supported languages](./v3-0-languages.md) included in the `dictionary` scope.
to	*Required parameter*. Specifies the language of the output text. The target language must be one of the [supported languages](./v3-0-languages.md) included in the `dictionary` scope.

Request headers include:

HEADERS	DESCRIPTION
Authentication header(s)	<i>Required request header.</i> See available options for authentication .
Content-Type	*Required request header*. Specifies the content type of the payload. Possible values are: `application/json`.
Content-Length	*Required request header*. The length of the request body.
X-ClientTraceId	*Optional*. A client-generated GUID to uniquely identify the request. You can omit this header if you include the trace ID in the query string using a query parameter named `ClientTraceId`.

Request body

The body of the request is a JSON array. Each array element is a JSON object with a string property named `Text`, which represents the term to lookup.

```
[
  { "Text": "fly" }
]
```

The following limitations apply:

- The array can have at most 10 elements.
- The text value of an array element cannot exceed 100 characters including spaces.

Response body

A successful response is a JSON array with one result for each string in the input array. A result object includes the following properties:

- **normalizedSource**: A string giving the normalized form of the source term. For example, if the request is "JOHN", the normalized form will be "john". The content of this field becomes the input to [lookup examples](#).
- **displaySource**: A string giving the source term in a form best suited for end-user display. For example, if the input is "JOHN", the display form will reflect the usual spelling of the name: "John".
- **translations**: A list of translations for the source term. Each element of the list is an object with the following properties:
 - **normalizedTarget**: A string giving the normalized form of this term in the target language. This value should be used as input to [lookup examples](#).
 - **displayTarget**: A string giving the term in the target language and in a form best suited for end-user display. Generally, this will only differ from the **normalizedTarget** in terms of capitalization. For example, a proper noun like "Juan" will have **normalizedTarget** = "juan" and **displayTarget** = "Juan".
 - **posTag**: A string associating this term with a part-of-speech tag.

TAG NAME	DESCRIPTION
ADJ	Adjectives
ADV	Adverbs
CONJ	Conjunctions
DET	Determiners
MODAL	Verbs
NOUN	Nouns
PREP	Prepositions
PRON	Pronouns
VERB	Verbs

TAG NAME	DESCRIPTION
OTHER	Other

As an implementation note, these tags were determined by part-of-speech tagging the English side, and then taking the most frequent tag for each source/target pair. So if people frequently translate a Spanish word to a different part-of-speech tag in English, tags may end up being wrong (with respect to the Spanish word).

- `confidence`: A value between 0.0 and 1.0 which represents the "confidence" (or perhaps more accurately, "probability in the training data") of that translation pair. The sum of confidence scores for one source word may or may not sum to 1.0.
- `prefixWord`: A string giving the word to display as a prefix of the translation. Currently, this is the gendered determiner of nouns, in languages that have gendered determiners. For example, the prefix of the Spanish word "mosca" is "la", since "mosca" is a feminine noun in Spanish. This is only dependent on the translation, and not on the source. If there is no prefix, it will be the empty string.
- `backTranslations`: A list of "back translations" of the target. For example, source words that the target can translate to. The list is guaranteed to contain the source word that was requested (e.g., if the source word being looked up is "fly", then it is guaranteed that "fly" will be in the `backTranslations` list). However, it is not guaranteed to be in the first position, and often will not be. Each element of the `backTranslations` list is an object described by the following properties:
 - `normalizedText`: A string giving the normalized form of the source term that is a back-translation of the target. This value should be used as input to [lookup examples](#).
 - `displayText`: A string giving the source term that is a back-translation of the target in a form best suited for end-user display.
 - `numExamples`: An integer representing the number of examples that are available for this translation pair. Actual examples must be retrieved with a separate call to [lookup examples](#). The number is mostly intended to facilitate display in a UX. For example, a user interface may add a hyperlink to the back-translation if the number of examples is greater than zero and show the back-translation as plain text if there are no examples. Note that the actual number of examples returned by a call to [lookup examples](#) may be less than `numExamples`, because additional filtering may be applied on the fly to remove "bad" examples.
 - `frequencyCount`: An integer representing the frequency of this translation pair in the data. The main purpose of this field is to provide a user interface with a means to sort back-translations so the most frequent terms are first.

NOTE

If the term being looked-up does not exist in the dictionary, the response is 200 (OK) but the `translations` list is an empty list.

Examples

This example shows how to lookup alternative translations in Spanish of the English term `fly`.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/dictionary/lookup?api-version=3.0&from=en&to=es" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json" -d "[{'Text': 'fly'}]"
```


The response body (abbreviated for clarity) is:

```
[
  {
    "normalizedSource": "fly",
    "displaySource": "fly",
    "translations": [
      {
        "normalizedTarget": "volar",
        "displayTarget": "volar",
        "posTag": "VERB",
        "confidence": 0.4081,
        "prefixWord": "",
        "backTranslations": [
          {"normalizedText": "fly", "displayText": "fly", "numExamples": 15, "frequencyCount": 4637},
          {"normalizedText": "flying", "displayText": "flying", "numExamples": 15, "frequencyCount": 1365},
          {"normalizedText": "blow", "displayText": "blow", "numExamples": 15, "frequencyCount": 503},
          {"normalizedText": "flight", "displayText": "flight", "numExamples": 15, "frequencyCount": 135}
        ]
      },
      {
        "normalizedTarget": "mosca",
        "displayTarget": "mosca",
        "posTag": "NOUN",
        "confidence": 0.2668,
        "prefixWord": "",
        "backTranslations": [
          {"normalizedText": "fly", "displayText": "fly", "numExamples": 15, "frequencyCount": 1697},
          {"normalizedText": "flyweight", "displayText": "flyweight", "numExamples": 0, "frequencyCount": 48},
          {"normalizedText": "flies", "displayText": "flies", "numExamples": 9, "frequencyCount": 34}
        ]
      }
    ],
    //
    // ...list abbreviated for documentation clarity
    //
  ]
}
```

This example shows what happens when the term being looked up does not exist for the valid dictionary pair.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/dictionary/lookup?api-version=3.0&from=en&to=es" -H "X-ClientTraceId: 875030C7-5380-40B8-8A03-63DACC69C11" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json" -d "[{'Text': 'fly123456'}]"
```

Since the term is not found in the dictionary, the response body includes an empty `translations` list.

```
[
  {
    "normalizedSource": "fly123456",
    "displaySource": "fly123456",
    "translations": []
  ]
}
```

Translator Text API 3.0: Dictionary Examples

11/8/2019 • 3 minutes to read • [Edit Online](#)

Provides examples that show how terms in the dictionary are used in context. This operation is used in tandem with [Dictionary lookup](#).

Request URL

Send a `POST` request to:

```
https://api.cognitive.microsofttranslator.com/dictionary/examples?api-version=3.0
```

Request parameters

Request parameters passed on the query string are:

QUERY PARAMETER	DESCRIPTION
api-version	*Required parameter*. Version of the API requested by the client. Value must be `3.0`.
from	*Required parameter*. Specifies the language of the input text. The source language must be one of the [supported languages](./v3-0-languages.md) included in the `dictionary` scope.
to	*Required parameter*. Specifies the language of the output text. The target language must be one of the [supported languages](./v3-0-languages.md) included in the `dictionary` scope.

Request headers include:

HEADERS	DESCRIPTION
Authentication header(s)	<i>Required request header.</i> See available options for authentication .
Content-Type	*Required request header*. Specifies the content type of the payload. Possible values are: `application/json`.
Content-Length	*Required request header*. The length of the request body.
X-ClientTraceId	*Optional*. A client-generated GUID to uniquely identify the request. You can omit this header if you include the trace ID in the query string using a query parameter named `ClientTraceId`.

Request body

The body of the request is a JSON array. Each array element is a JSON object with the following properties:

- `Text`: A string specifying the term to lookup. This should be the value of a `normalizedText` field from the

back-translations of a previous [Dictionary lookup](#) request. It can also be the value of the `normalizedSource` field.

- `Translation`: A string specifying the translated text previously returned by the [Dictionary lookup](#) operation. This should be the value from the `normalizedTarget` field in the `translations` list of the [Dictionary lookup](#) response. The service will return examples for the specific source-target word-pair.

An example is:

```
[
  { "Text": "fly", "Translation": "volar" }
]
```

The following limitations apply:

- The array can have at most 10 elements.
- The text value of an array element cannot exceed 100 characters including spaces.

Response body

A successful response is a JSON array with one result for each string in the input array. A result object includes the following properties:

- `normalizedSource`: A string giving the normalized form of the source term. Generally, this should be identical to the value of the `Text` field at the matching list index in the body of the request.
- `normalizedTarget`: A string giving the normalized form of the target term. Generally, this should be identical to the value of the `Translation` field at the matching list index in the body of the request.
- `examples`: A list of examples for the (source term, target term) pair. Each element of the list is an object with the following properties:
 - `sourcePrefix`: The string to concatenate *before* the value of `sourceTerm` to form a complete example. Do not add a space character, since it is already there when it should be. This value may be an empty string.
 - `sourceTerm`: A string equal to the actual term looked up. The string is added with `sourcePrefix` and `sourceSuffix` to form the complete example. Its value is separated so it can be marked in a user interface, e.g., by bolding it.
 - `sourceSuffix`: The string to concatenate *after* the value of `sourceTerm` to form a complete example. Do not add a space character, since it is already there when it should be. This value may be an empty string.
 - `targetPrefix`: A string similar to `sourcePrefix` but for the target.
 - `targetTerm`: A string similar to `sourceTerm` but for the target.
 - `targetSuffix`: A string similar to `sourceSuffix` but for the target.

NOTE

If there are no examples in the dictionary, the response is 200 (OK) but the `examples` list is an empty list.

Examples

This example shows how to lookup examples for the pair made up of the English term `fly` and its Spanish translation `volar`.

```
curl -X POST "https://api.cognitive.microsofttranslator.com/dictionary/examples?api-version=3.0&from=en&to=es" -H "Ocp-Apim-Subscription-Key: <client-secret>" -H "Content-Type: application/json" -d "[{'Text': 'fly', 'Translation': 'volar'}]"
```

The response body (abbreviated for clarity) is:

```
[
  {
    "normalizedSource": "fly",
    "normalizedTarget": "volar",
    "examples": [
      {
        "sourcePrefix": "They need machines to ",
        "sourceTerm": "fly",
        "sourceSuffix": ".",
        "targetPrefix": "Necesitan máquinas para ",
        "targetTerm": "volar",
        "targetSuffix": "."
      },
      {
        "sourcePrefix": "That should really ",
        "sourceTerm": "fly",
        "sourceSuffix": ".",
        "targetPrefix": "Eso realmente debe ",
        "targetTerm": "volar",
        "targetSuffix": "."
      },
      //
      // ...list abbreviated for documentation clarity
      //
    ]
  }
]
```

Translator Text API v2.0

10/9/2019 • 30 minutes to read • [Edit Online](#)

IMPORTANT

This version of the Translator Text API has been deprecated. [View documentation for version 3 of the Translator Text API.](#)

Version 2 of the Translator Text API can be seamlessly integrated into your apps, websites, tools, or other solutions to provide multilanguage user experiences. You can use it on any hardware platform and with any operating system to perform language translation and other language-related tasks, like text-language detection and text to speech, according to industry standards. For more information, see [Translator Text API](#).

Getting started

To access the Translator Text API, you need to [sign up for Microsoft Azure](#).

Authentication

All calls to the Translator Text API require a subscription key for authentication. The API supports three methods of authentication:

- An access token. Use the subscription key to create an access token by making a POST request to the authentication service. See the token service documentation for details. Pass the access token to the Translator service by using the `Authorization` header or the `access_token` query parameter. The access token is valid for 10 minutes. Obtain a new access token every 10 minutes, and keep using the same access token for repeated requests during the 10 minutes.
- A subscription key used directly. Pass your subscription key as a value in the `Ocp-Apim-Subscription-Key` header included with your request to the Translator Text API. When you use the subscription key directly, you don't have to call the token authentication service to create an access token.
- An [Azure Cognitive Services multi-service subscription](#). This method allows you to use a single secret key to authenticate requests for multiple services. When you use a multi-service secret key, you need to include two authentication headers with your request. The first header passes the secret key. The second header specifies the region associated with your subscription:
 - `Ocp-Apim-Subscription-Key`
 - `Ocp-Apim-Subscription-Region`

The region is required for the multi-service Text API subscription. The region you select is the only region you can use for text translation when you use the multi-service subscription key. It needs to be the same region you selected when you signed up for your multi-service subscription on the Azure portal.

The available regions are `australiaeast`, `brazilsouth`, `canadacentral`, `centralindia`, `centraluseuap`, `eastasia`, `eastus`, `eastus2`, `japaneast`, `northeurope`, `southcentralus`, `southeastasia`, `uksouth`, `westcentralus`, `westeurope`, `westus`, and `westus2`.

Your subscription key and access token are secrets that should be hidden from view.

Profanity handling

Normally, the Translator service will retain profanity that's present in the source. The degree of profanity and the

context that makes words profane differ according to culture. So the degree of profanity in the target language could be increased or reduced.

If you want to prevent profanity in the translation even when it's in the source text, you can use the profanity filtering option for the methods that support it. The option allows you to choose whether you want to see profanity deleted or marked with appropriate tags, or whether you want to allow the profanity in the target. The accepted values of `ProfanityAction` are `NoAction` (default), `Marked`, and `Deleted`.

PROFANITYACTION	ACTION	EXAMPLE SOURCE (JAPANESE)	EXAMPLE TRANSLATION (ENGLISH)
NoAction	Default. Same as not setting the option. Profanity will pass from source to target.	彼はジャッカスです。	He is a jackass.
Marked	Profane words will be surrounded by XML tags <code><profanity></code> and <code></profanity></code> .	彼はジャッカスです。	He is a <code><profanity>jackass</profanity></code> .
Deleted	Profane words will be removed from the output without replacement.	彼はジャッカスです。	He is a.

Excluding content from translation

When you translate content with tags, like HTML (`contentType=text/html`), it's sometimes useful to exclude specific content from the translation. You can use the attribute `class=nottranslate` to specify content that should remain in its original language. In the following example, the content in the first `div` element won't be translated, but the content in the second `div` element will be translated.

```
<div class="nottranslate">This will not be translated.</div>
<div>This will be translated. </div>
```

GET /Translate

Implementation notes

Translates a text string from one language to another.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/Translate`.

Return value: A string that represents the translated text.

If you previously used `AddTranslation` or `AddTranslationArray` to enter a translation with a rating of 5 or higher for the same source sentence, `Translate` returns only the top choice that's available to your system. "Same source sentence" means exactly the same (100% matching), except for capitalization, white space, tag values, and punctuation at the end of a sentence. If no rating is stored with a rating of 5 or above, the returned result will be the automatic translation by Microsoft Translator.

Response class (status 200)

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> . .	query	string
text	(empty)	Required. A string that represents the text to translate. The text can't contain more than 10,000 characters.	query	string
from	(empty)	Optional. A string that represents the language code of the text being translated. For example, en for English.	query	string
to	(empty)	Required. A string that represents the code of the language to translate the text into.	query	string
contentType	(empty)	Optional. The format of the text being translated. Supported formats are <code>text/plain</code> (default) and <code>text/html</code> . Any HTML elements need to be well-formed, complete elements.	query	string
category	(empty)	Optional. A string that contains the category (domain) of the translation. The default is <code>general</code> .	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> . .	header	string

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

POST /TranslateArray

Implementation notes

Retrieves translations for multiple source texts.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/TranslateArray`.

Here's the format of the request body:

```
<TranslateArrayRequest>
  <AppId />
  <From>language-code</From>
  <Options>
    <Category xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2" >string-value</Category>
    <ContentType
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">text/plain</ContentType>
    <ReservedFlags xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2" />
    <State xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2" >int-value</State>
    <Uri xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2" >string-value</Uri>
    <User xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2" >string-value</User>
  </Options>
  <Texts>
    <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">string-value</string>
    <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">string-value</string>
  </Texts>
  <To>language-code</To>
</TranslateArrayRequest>
```

These elements are in `TranslateArrayRequest`:

- `AppId`: Required. If the `Authorization` or `Ocp-Apim-Subscription-Key` header is used, leave the `AppId` field empty. Otherwise, include a string that contains `"Bearer" + " " + "access_token"`.
- `From`: Optional. A string that represents the language code of the text being translated. If this field is left empty,

the response will include the result of automatic language detection.

- **Options**: Optional. An **Options** object that contains the following values. They're all optional and default to the most common settings. Specified elements must be listed in alphabetical order.
 - **Category**: A string that contains the category (domain) of the translation. The default is **general**.
 - **ContentType**: The format of the text being translated. The supported formats are **text/plain** (default), **text/xml**, and **text/html**. Any HTML elements need to be well-formed, complete elements.
 - **ProfanityAction**: Specifies how profanities are handled, as explained earlier. Accepted values are **NoAction** (default), **Marked**, and **Deleted**.
 - **State**: User state to help correlate the request and response. The same content will be returned in the response.
 - **Uri**: Filter results by this URI. Default: **all**.
 - **User**: Filter results by this user. Default: **all**.
- **Texts**: Required. An array that contains the text for translation. All strings must be in the same language. The total of all text to be translated can't exceed 10,000 characters. The maximum number of array elements is 2,000.
- **To**: Required. A string that represents the code of the language to translate the text into.

You can omit optional elements. Elements that are direct children of **TranslateArrayRequest** must be listed in alphabetical order.

The **TranslateArray** method accepts **application/xml** or **text/xml** for **Content-Type**.

Return value: A **TranslateArrayResponse** array. Each **TranslateArrayResponse** has these elements:

- **Error**: Indicates an error if one occurs. Otherwise set to null.
- **OriginalSentenceLengths**: An array of integers that indicates the length of each sentence in the source text. The length of the array indicates the number of sentences.
- **TranslatedText**: The translated text.
- **TranslatedSentenceLengths**: An array of integers that indicates the length of each sentence in the translated text. The length of the array indicates the number of sentences.
- **State**: User state to help correlate the request and response. Returns the same content as the request.

Here's the format of the response body:

```
<ArrayOfTranslateArrayResponse xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2"
  xmlns:i="https://www.w3.org/2001/XMLSchema-instance">
  <TranslateArrayResponse>
    <From>language-code</From>
    <OriginalTextSentenceLengths xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
      <a:int>int-value</a:int>
    </OriginalTextSentenceLengths>
    <State/>
    <TranslatedText>string-value</TranslatedText>
    <TranslatedTextSentenceLengths xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
      <a:int>int-value</a:int>
    </TranslatedTextSentenceLengths>
  </TranslateArrayResponse>
</ArrayOfTranslateArrayResponse>
```

Response class (status 200)

A successful response includes an array of **TranslateArrayResponse** arrays in the format described earlier.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response. Common errors include: <ul style="list-style-type: none">• Array element cannot be empty.• Invalid category.• From language is invalid.• To language is invalid.• The request contains too many elements.• The From language is not supported.• The To language is not supported.• Translate Request has too much data.• HTML is not in a correct format.• Too many strings were passed in the Translate Request.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

POST /GetLanguageNames

Implementation notes

Retrieves friendly names for the languages passed in as the parameter `languageCodes` , localized into the passed `locale` language.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/GetLanguageNames` .

The request body includes a string array that represents the ISO 639-1 language codes for which to retrieve the friendly names. Here's an example:

```
<ArrayOfstring xmlns:i="https://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
  <string>zh</string>
  <string>en</string>
</ArrayOfstring>
```

Return value: A string array that contains language names supported by the Translator service, localized into the requested language.

Response class (status 200)

A string array that contains languages names supported by the Translator service, localized into the requested language.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
locale	(empty)	Required. A string that represents one of the following, used to localize the language names: <ul style="list-style-type: none"> The combination of an ISO 639 two-letter lowercase culture code associated with a language and an ISO 3166 two-letter uppercase subculture code. An ISO 639 lowercase culture code by itself. 	query	string

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

GET /GetLanguagesForTranslate

Implementation notes

Gets a list of language codes that represent languages supported by the Translation service. `Translate` and `TranslateArray` can translate between any two of these languages.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/GetLanguagesForTranslate` .

Return value: A string array that contains the language codes supported by the Translator service.

Response class (status 200)

A string array that contains the language codes supported by the Translator service.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
-----------	-------	-------------	----------------	-----------

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

GET /GetLanguagesForSpeak

Implementation notes

Retrieves the languages available for speech synthesis.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/GetLanguagesForSpeak`.

Return value: A string array that contains the language codes supported for speech synthesis by the Translator service.

Response class (status 200)

A string array that contains the language codes supported for speech synthesis by the Translator service.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

GET /Speak

Implementation notes

Returns a WAV or MP3 stream of the passed-in text, spoken in the desired language.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/Speak`.

Return value: A WAV or MP3 stream of the passed-in text, spoken in the desired language.

Response class (status 200)

binary

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
text	(empty)	Required. A string that contains one or more sentences to be spoken for the stream, in the specified language. The text must not exceed 2,000 characters.	query	string
language	(empty)	Required. A string that represents the supported language code of the language in which to speak the text. The code must be one of the codes returned by the method <code>GetLanguagesForSpeak</code> .	query	string
format	(empty)	Optional. A string that specifies the content-type ID. Currently, <code>audio/wav</code> and <code>audio/mp3</code> are available. The default value is <code>audio/wav</code> .	query	string

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
options	(empty)	<p>Optional. A string that specifies properties of the synthesized speech:</p> <ul style="list-style-type: none"> <code>MaxQuality</code> and <code>MinSize</code> specify the quality of the audio signal. <code>MaxQuality</code> provides the highest quality. <code>MinSize</code> provides the smallest file size. The default is <code>MinSize</code>. <code>female</code> and <code>male</code> specify the desired gender of the voice. The default is <code>female</code>. Use the vertical bar (<code> </code>) to include multiple options. For example, <code>MaxQuality Male</code>. 	query	string
Authorization	(empty)	<p>Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token:</p> <pre>"Bearer" + " " + "access_token"</pre>	header	string
Ocp-Apim-Subscription-Key	(empty)	<p>Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.</p>	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.

HTTP STATUS CODE	REASON
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

GET /Detect

Implementation notes

Identifies the language of a section of text.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/Detect`.

Return value: A string that contains a two-character language code for the text.

Response class (status 200)

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
text	(empty)	Required. A string that contains text whose language is to be identified. The text must not exceed 10,000 characters.	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

POST /DetectArray

Implementation notes

Identifies the languages in an array of strings. Independently detects the language of each individual array element and returns a result for each row of the array.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/DetectArray`.

Here's the format of the request body:

```
<ArrayOfstring xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
  <string>string-value-1</string>
  <string>string-value-2</string>
</ArrayOfstring>
```

The text can't exceed 10,000 characters.

Return value: A string array that contains a two-character language code for each row in the input array.

Here's the format of the response body:

```
<ArrayOfstring xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays"
xmlns:i="https://www.w3.org/2001/XMLSchema-instance">
  <string>language-code-1</string>
  <string>language-code-2</string>
</ArrayOfstring>
```

Response class (status 200)

`DetectArray` was successful. Returns a string array that contains a two-character language code for each row of the input array.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> . .	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> . .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> . .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

GET /AddTranslation

Implementation notes

IMPORTANT

Deprecation note: After January 31, 2018, this method won't accept new sentence submissions. You'll get an error message. Please see the announcement about changes to the Collaborative Translation Framework (CTF).

Adds a translation to the translation memory.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/AddTranslation`.

Response class (status 200)

string

Response content type: application: xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
originalText	(empty)	Required. A string that contains the text to translate. The maximum length of the string is 1,000 characters.	query	string
translatedText	(empty)	Required. A string that contains text translated into the target language. The maximum length of the string is 2,000 characters.	query	string
from	(empty)	Required. A string that represents the language code of the original language of the text. For example, en for English and de for German.	query	string
to	(empty)	Required. A string that represents the language code of the language to translate the text into.	query	string

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
rating	(empty)	Optional. An integer that represents the quality rating for the string. The value is between -10 and 10. The default is 1.	query	integer
contentType	(empty)	Optional. The format of the text being translated. The supported formats are <code>text/plain</code> and <code>text/html</code> . Any HTML elements need to be well-formed, complete elements.	query	string
category	(empty)	Optional. A string that contains the category (domain) of the translation. The default is <code>general</code> .	query	string
user	(empty)	Required. A string that's used to track the originator of the submission.	query	string
uri	(empty)	Optional. A string that contains the content location of the translation.	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.

HTTP STATUS CODE	REASON
410	<code>AddTranslation</code> is no longer supported.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

POST /AddTranslationArray

Implementation notes

IMPORTANT

Deprecation note: After January 31, 2018, this method won't accept new sentence submissions. You'll get an error message. Please see the announcement about changes to the Collaborative Translation Framework (CTF).

Adds an array of translations to translation memory. This method is an array version of `AddTranslation`.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/AddTranslationArray`.

Here's the format of the request body:

```
<AddtranslationsRequest>
  <AppId></AppId>
  <From>A string containing the language code of the source language</From>
  <Options>
    <Category xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">string-
value</Category>
    <ContentType
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">text/plain</ContentType>
    <User xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">string-value</User>
    <Uri xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">string-value</Uri>
  </Options>
  <To>A string containing the language code of the target language</To>
  <Translations>
    <Translation xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">
      <OriginalText>string-value</OriginalText>
      <Rating>int-value</Rating>
      <Sequence>int-value</Sequence>
      <TranslatedText>string-value</TranslatedText>
    </Translation>
  </Translations>
</AddtranslationsRequest>
```

These elements are in `AddtranslationsRequest`:

- `AppId`: Required. If the `Authorization` or `Ocp-Apim-Subscription-Key` header is used, leave the `AppId` field empty. Otherwise, include a string that contains "Bearer" + " " + "access_token".
- `From`: Required. A string that contains the language code of the source language. Must be one of the languages returned by the `GetLanguagesForTranslate` method.
- `To`: Required. A string that contains the language code of the target language. Must be one of the languages returned by the `GetLanguagesForTranslate` method.

- **Translations** : Required. An array of translations to add to translation memory. Each translation must contain **OriginalText** , **TranslatedText** , and **Rating** . The maximum size of each **OriginalText** and **TranslatedText** is 1,000 characters. The total of all **OriginalText** and **TranslatedText** elements can't exceed 10,000 characters. The maximum number of array elements is 100.
- **Options** : Required. A set of options, including **Category** , **ContentType** , **Uri** , and **User** . **User** is required. **Category** , **ContentType** , and **Uri** are optional. Specified elements must be listed in alphabetical order.

Response class (status 200)

AddTranslationArray method succeeded.

After January 31, 2018, sentence submissions won't be accepted. The service will respond with error code 410.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
Authorization	(empty)	Required if both the appid field and the Ocp-Apim-Subscription-Key header are left empty. Authorization token: "Bearer" + " " + "access_token" .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the appid field and the Authorization header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
410	AddTranslation is no longer supported.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header X-MS-Trans-Info .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

GET /BreakSentences

Implementation notes

Breaks a section of text into sentences and returns an array that contains the lengths of each sentence.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/BreakSentences`.

Return value: An array of integers that represents the lengths of the sentences. The length of the array represents the number of sentences. The values represent the length of each sentence.

Response class (status 200)

An array of integers that represents the lengths of the sentences. The length of the array represents the number of sentences. The values represent the length of each sentence.

integer

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
text	(empty)	Required. A string that represents the text to split into sentences. The maximum size of the text is 10,000 characters.	query	string
language	(empty)	Required. A string that represents the language code of the input text.	query	string
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

POST /GetTranslations

Implementation notes

Retrieves an array of translations for a given language pair from the store and the MT engine. `GetTranslations` differs from `Translate` in that it returns all available translations.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/GetTranslations`.

The body of the request includes the optional `TranslateOptions` object, which has this format:

```
<TranslateOptions xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">
  <Category>string-value</Category>
  <ContentType>text/plain</ContentType>
  <ReservedFlags></ReservedFlags>
  <State>int-value</State>
  <Uri>string-value</Uri>
  <User>string-value</User>
</TranslateOptions>
```

The `TranslateOptions` object contains the values in the following list. They're all optional and default to the most common settings. Specified elements must be listed in alphabetical order.

- `category`: A string that contains the category (domain) of the translation. The default is `general`.
- `ContentType`: The only supported option, and the default, is `text/plain`.
- `IncludeMultipleMTAlternatives`: A Boolean flag to specify whether more than one alternative should be returned from the MT engine. Valid values are `true` and `false` (case-sensitive). The default is `false`, which returns only one alternative. Setting the flag to `true` enables the creation of artificial alternatives, fully integrated with the Collaborative Translation Framework (CTF). The feature enables returning alternatives for sentences that have no translations in CTF by adding artificial alternatives from the *n*-best list of the decoder.
 - Ratings. The ratings are applied like this:
 - The best automatic translation has a rating of 5.
 - The alternatives from CTF reflect the authority of the reviewer. They range from -10 to +10.
 - The automatically generated (*n*-best) translation alternatives have a rating of 0 and a match degree of 100.
 - Number of alternatives. The number of returned alternatives can be as high as the value specified in `maxTranslations`, but it can be lower.
 - Language pairs. This functionality isn't available for translations between Simplified Chinese and Traditional Chinese, in either direction. It is available for all other language pairs supported by Microsoft

Translator.

- `State`: User state to help correlate the request and response. The same content will be returned in the response.
- `Uri`: Filter results by this URI. If no value is set, the default is `all`.
- `User`: Filter results by this user. If no value is set, the default is `all`.

Request `Content-Type` should be `text/xml`.

Return value: Here's the format of the response:

```
<GetTranslationsResponse xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2"
  xmlns:i="https://www.w3.org/2001/XMLSchema-instance">
  <From>Two character language code</From>
  <State/>
  <Translations>
    <TranslationMatch>
      <Count>int-value</Count>
      <MatchDegree>int-value</MatchDegree>
      <MatchedOriginalText/>
      <Rating>int value</Rating>
      <TranslatedText>string-value</TranslatedText>
    </TranslationMatch>
  </Translations>
</GetTranslationsResponse>
```

This response includes a `GetTranslationsResponse` element that contains the following values:

- `Translations`: An array of the matches found, stored in `TranslationMatch` objects (described in the following section). The translations might include slight variants of the original text (fuzzy matching). The translations will be sorted: 100% matches first, fuzzy matches next.
- `From`: If the method doesn't specify a `From` language, this value will come from automatic language detection. Otherwise, it will be the specified `From` language.
- `State`: User state to help correlate the request and response. Contains the value supplied in the `TranslateOptions` parameter.

The `TranslationMatch` object consists of these values:

- `Error`: The error code, if an error occurs for a specific input string. Otherwise, this field is empty.
- `MatchDegree`: Indicates how closely the input text matches the original text found in the store. The system matches input sentences against the store, including inexact matches. The value returned ranges from 0 to 100, where 0 is no similarity and 100 is an exact, case-sensitive match.
- `MatchedOriginalText`: Original text that was matched for this result. This value is returned only if the matched original text was different from the input text. It's used to return the source text of a fuzzy match. This value isn't returned for Microsoft Translator results.
- `Rating`: Indicates the authority of the person making the quality decision. Machine Translation results have a rating of 5. Anonymously provided translations generally have a rating from 1 through 4. Authoritatively provided translations will generally have a rating from 6 through 10.
- `Count`: The number of times this translation with this rating has been selected. The value is 0 for the automatically translated response.
- `TranslatedText`: The translated text.

Response class (status 200)

A `GetTranslationsResponse` object in the format described previously.

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
appid	(empty)	Required. If the <code>Authorization</code> or <code>Ocp-Apim-Subscription-Key</code> header is used, leave the <code>appid</code> field empty. Otherwise, include a string that contains <code>"Bearer" + " " + "access_token"</code> .	query	string
text	(empty)	Required. A string that represents the text to translate. The maximum size of the text is 10,000 characters.	query	string
from	(empty)	Required. A string that represents the language code of the text being translated.	query	string
to	(empty)	Required. A string that represents the language code of the language to translate the text into.	query	string
maxTranslations	(empty)	Required. An integer that represents the maximum number of translations to return.	query	integer
Authorization	(empty)	Required if both the <code>appid</code> field and the <code>Ocp-Apim-Subscription-Key</code> header are left empty. Authorization token: <code>"Bearer" + " " + "access_token"</code> .	string	header
Ocp-Apim-Subscription-Key	(empty)	Required if both the <code>appid</code> field and the <code>Authorization</code> header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

POST /GetTranslationsArray

Implementation notes

Retrieves multiple translation candidates for multiple source texts.

The request URI is `https://api.microsofttranslator.com/V2/Http.svc/GetTranslationsArray`.

Here's the format of the request body:

```
<GetTranslationsArrayRequest>
  <AppId></AppId>
  <From>language-code</From>
  <MaxTranslations>int-value</MaxTranslations>
  <Options>
    <Category xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">string-
value</Category>
    <ContentType
xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">text/plain</ContentType>
    <ReservedFlags xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2"/>
    <State xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">int-value</State>
    <Uri xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">string-value</Uri>
    <User xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2">string-value</User>
  </Options>
  <Texts>
    <string xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays">string-value</string>
  </Texts>
  <To>language-code</To>
</GetTranslationsArrayRequest>
```

`GetTranslationsArrayRequest` includes these elements:

- `AppId`: Required. If the `Authorization` header is used, leave the `AppId` field empty. Otherwise, include a string that contains `"Bearer" + " " + "access_token"`.
- `From`: Required. A string that represents the language code of the text being translated.
- `MaxTranslations`: Required. An integer that represents the maximum number of translations to return.
- `Options`: Optional. An `options` object that contains the following values. They're all optional and default to the most common settings. Specified elements must be listed in alphabetical order.
 - `Category`: A string that contains the category (domain) of the translation. The default is `general`.
 - `ContentType`: The only supported option, and the default, is `text/plain`.
 - `IncludeMultipleMTAlternatives`: A Boolean flag to specify whether more than one alternative should be returned from the MT engine. Valid values are `true` and `false` (case-sensitive). The default is `false`.

which returns only one alternative. Setting the flag to `true` enables generation of artificial alternatives in translation, fully integrated with the Collaborative Translations Framework (CTF). The feature enables returning alternatives for sentences that have no alternatives in CTF by adding artificial alternatives from the *n*-best list of the decoder.

- Ratings The ratings are applied like this:
 - The best automatic translation has a rating of 5.
 - The alternatives from CTF reflect the authority of the reviewer. They range from -10 to +10.
 - The automatically generated (*n*-best) translation alternatives have a rating of 0 and a match degree of 100.
 - Number of alternatives. The number of returned alternatives can be as high as the value specified in `maxTranslations`, but it can be lower.
 - Language pairs. This functionality isn't available for translations between Simplified Chinese and Traditional Chinese, in either direction. It is available for all other language pairs supported by Microsoft Translator.
- `state`: User state to help correlate the request and response. The same content will be returned in the response.
 - `Uri`: Filter results by this URI. If no value is set, the default is `all`.
 - `User`: Filter results by this user. If no value is set, the default is `all`.
 - `Texts`: Required. An array that contains the text for translation. All strings must be in the same language. The total of all text to be translated can't exceed 10,000 characters. The maximum number of array elements is 10.
 - `To`: Required. A string that represents the language code of the language to translate the text into.

You can omit optional elements. Elements that are direct children of `GetTranslationsArrayRequest` must be listed in alphabetical order.

Request `Content-Type` should be `text/xml`.

Return value: Here's the format of the response:

```
<ArrayOfGetTranslationsResponse xmlns="http://schemas.datacontract.org/2004/07/Microsoft.MT.Web.Service.V2"
xmlns:i="https://www.w3.org/2001/XMLSchema-instance">
  <GetTranslationsResponse>
    <From>language-code</From>
    <State/>
    <Translations>
      <TranslationMatch>
        <Count>int-value</Count>
        <MatchDegree>int-value</MatchDegree>
        <MatchedOriginalText>string-value</MatchedOriginalText>
        <Rating>int-value</Rating>
        <TranslatedText>string-value</TranslatedText>
      </TranslationMatch>
      <TranslationMatch>
        <Count>int-value</Count>
        <MatchDegree>int-value</MatchDegree>
        <MatchedOriginalText/>
        <Rating>int-value</Rating>
        <TranslatedText>string-value</TranslatedText>
      </TranslationMatch>
    </Translations>
  </GetTranslationsResponse>
</ArrayOfGetTranslationsResponse>
```

Each `GetTranslationsResponse` element contains these values:

- `Translations`: An array of the matches found, stored in `TranslationMatch` objects (described in the following section). The translations might include slight variants of the original text (fuzzy matching). The translations will

be sorted: 100% matches first, fuzzy matches next.

- **From** : If the method doesn't specify a **From** language, this value will come from automatic language detection. Otherwise, it will be the specified **From** language.
- **state** : User state to help correlate the request and response. Contains the value supplied in the **TranslateOptions** parameter.

The **TranslationMatch** object contains the following values:

- **Error** : The error code, if an error occurs for a specific input string. Otherwise, this field is empty.
- **MatchDegree** : Indicates how closely the input text matches the original text found in the store. The system matches input sentences against the store, including inexact matches. The value returned ranges from 0 to 100, where 0 is no similarity and 100 is an exact, case-sensitive match.
- **MatchedOriginalText** : Original text that was matched for this result. This value is returned only if the matched original text was different from the input text. It's used to return the source text of a fuzzy match. This value isn't returned for Microsoft Translator results.
- **Rating** : Indicates the authority of the person making the quality decision. Machine Translation results have a rating of 5. Anonymously provided translations generally have a rating from 1 through 4. Authoritatively provided translations generally have a rating from 6 through 10.
- **Count** : The number of times this translation with this rating has been selected. The value is 0 for the automatically translated response.
- **TranslatedText** : The translated text.

Response class (status 200)

string

Response content type: application/xml

Parameters

PARAMETER	VALUE	DESCRIPTION	PARAMETER TYPE	DATA TYPE
Authorization	(empty)	Required if both the appid field and the Ocp-Apim-Subscription-Key header are left empty. Authorization token: "Bearer" + " " + "access_token" .	header	string
Ocp-Apim-Subscription-Key	(empty)	Required if both the appid field and the Authorization header are left empty.	header	string

Response messages

HTTP STATUS CODE	REASON
400	Bad request. Check input parameters and the detailed error response.
401	Invalid credentials.

HTTP STATUS CODE	REASON
500	Server error. If the error persists, let us know. Please provide us with the approximate date & time of the request and with the request ID included in the response header <code>X-MS-Trans-Info</code> .
503	Service temporarily unavailable. Please retry and let us know if the error persists.

Next steps

[Migrate to Translator Text API v3](#)

How to use the TransformText method

7/28/2019 • 3 minutes to read • [Edit Online](#)

NOTE

This method is deprecated. It is not available in V3.0 of the Translator Text API.

The TransformText method is a text normalization function for social media, which returns a normalized form of the input. The method can be used as a preprocessing step in machine translation or other applications which expect clean input text that is not typically found in social media or user-generated content. The function currently works only with English input.

The method is a RESTful service using GET over HTTP. It supports simple XML and JSON serialization.

Parameters

PARAMETER	DESCRIPTION
Authorization header	Required HTTP header used to identify the application. Use key: "Authorization" and value: "Bearer" + " " + access token. For details, go here .
language	Required A string representing the language code. This parameter supports only English with en as the language name.
category	Optional A string containing the category or domain of the translation. This parameter supports only the default option general .
sentence	Required A sentence that you want to correct.

Return value

The return value provides the transformed sentence.

```
GetTranslationsResponse Microsoft.Translator.GetTranslations(appId, text, from, to, maxTranslations, options);
TransformTextResponse
{
    int ec;           // A positive number representing an error condition
    string em;        // A descriptive error message
    string sentence;   // transformed text
}
```

Example

```
using System;
using System.Text;
using System.Net;
using System.IO;
//Requires System.Runtime.Serialization assembly to be referenced
```



```

//Requires System.Web assembly to be referenced
using System.Runtime.Serialization.Json;
using System.Runtime.Serialization;
//Requires System.Web assembly to be referenced
using System.Web;
using System.Media;
using System.Threading;
namespace MicrosoftTranslatorSdk.HttpSamples
{
    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 (https://msdn.microsoft.com/library/hh454950.aspx)
            AdmAuthentication admAuth = new AdmAuthentication("clientId", "client secret");

            try
            {
                admToken = admAuth.GetAccessToken();
                // Create a header with the access_token property of the returned token
                headerValue = "Bearer " + admToken.access_token;
                TransformTextMethod(headerValue);
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey(true);
            }
            catch (WebException e)
            {
                ProcessWebException(e);
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey(true);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("Press any key to continue...");
                Console.ReadKey(true);
            }
        }
        private static void TransformTextMethod(string authToken)
        {
            Console.WriteLine("Enter Text to transform: (e.g Dis is 2 strange i juss wanna go home soooooooooon)");
            string textToTransform = Console.ReadLine();
            string language = "en";
            string domain = "general";
            //Keep appId parameter blank as we are sending access token in authorization header.
            string url = string.Format("https://api.microsofttranslator.com/V3/json/TransformText?sentence={0}&category={1}&language={2}", textToTransform, domain, language); ;
            TransformTextResponse transformTextResponse= new TransformTextResponse();
            HttpRequest httpWebRequest = (HttpRequest)WebRequest.Create(url);
            httpWebRequest.Headers.Add("Authorization", authToken);
            using (WebResponse response = httpWebRequest.GetResponse())
            {
                using( StreamReader sr = new StreamReader(response.GetResponseStream()))
                {
                    using (MemoryStream ms = new MemoryStream(Encoding.UTF8.GetBytes(sr.ReadToEnd())))
                    {
                        DataContractJsonSerializer serializer = new
DataContractJsonSerializer(typeof(TransformTextResponse));
                        //Get deserialized TransformTextResponse object from JSON stream
                        transformTextResponse = (TransformTextResponse)serializer.ReadObject(ms);
                        Console.WriteLine("Transformed sentence: {0}", transformTextResponse.sentence);
                    }
                }
            }
        }
        private static void ProcessWebException(WebException e)

```

```

private static void ProcessWebException(WebException e)
{
    Console.WriteLine("{0}", e.ToString());
    // Obtain detailed error information
    string strResponse = string.Empty;
    using (HttpWebResponse response = (HttpWebResponse)e.Response)
    {
        using (Stream responseStream = response.GetResponseStream())
        {
            using (StreamReader sr = new StreamReader(responseStream, System.Text.Encoding.ASCII))
            {
                strResponse = sr.ReadToEnd();
            }
        }
    }
    Console.WriteLine("Http status code={0}, error message={1}", e.Status, strResponse);
}
}
[DataContract]
public class TransformTextResponse
{
    [DataMember]
    public int ec { get; set; }
    [DataMember]
    public string em { get; set; }
    [DataMember]
    public string sentence { get; set; }
}
[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;
    private AdmAccessToken token;
    private Timer accessTokenRenewer;
    //Access token expires every 10 minutes. Renew it every 9 minutes only.
    private const int RefreshTokenDuration = 9;
    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));
        this.token = HttpPost(DatamarketAccessUri, this.request);
        //renew the token every specified minutes
        accessTokenRenewer = new Timer(new TimerCallback(OnTokenExpiredCallback), this,
TimeSpan.FromMinutes(RefreshTokenDuration), TimeSpan.FromMilliseconds(-1));
    }
    public AdmAccessToken GetAccessToken()
    {
        return this.token;
    }
    private void RenewAccessToken()

```

```

        {
            AdmAccessToken newAccessToken = HttpPost(DatamarketAccessUri, this.request);
            this.token = newAccessToken;
            Console.WriteLine(string.Format("Renewed token for user: {0} is: {1}", this.clientId,
this.token.access_token));
        }
        private void OnTokenExpiredCallback(object stateInfo)
        {
            try
            {
                RenewAccessToken();
            }
            catch (Exception ex)
            {
                Console.WriteLine(string.Format("Failed renewing access token. Details: {0}", ex.Message));
            }
            finally
            {
                try
                {
                    accessTokenRenewer.Change(TimeSpan.FromMinutes(RefreshTokenDuration),
TimeSpan.FromMilliseconds(-1));
                }
                catch (Exception ex)
                {
                    Console.WriteLine(string.Format("Failed to reschedule the timer to renew access token.
Details: {0}", ex.Message));
                }
            }
        }
        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;
            }
        }
    }
}

```

How to use Collaborative Translation Framework (CTF) reporting

7/28/2019 • 6 minutes to read • [Edit Online](#)

NOTE

This method is deprecated. It is not available in V3.0 of the Translator Text API.

The Collaborative Translations Framework (CTF), previously available for V2.0 of the Translator Text API, was deprecated as of February 1, 2018. The `AddTranslation` and `AddTranslationArray` functions let users enable corrections through the Collaborative Translation Framework. After January 31, 2018, these two functions did not accept new sentence submissions, and users receive an error message. These functions were retired and will not be replaced.

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 <https://api.microsofttranslator.com/v2/beta/ctfreporting.svc>

Methods

NAME	DESCRIPTION
<code>GetUserTranslationCounts</code> Method	Get counts of the translations that are created by the user.
<code>GetUserTranslations</code> Method	Retrieves the translations that are created by the user.

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.

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

```
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.
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 the 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.

PARAMETER	DESCRIPTION
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
ArgumentOutOfRangeException	The parameter ' maxDateUtc ' must be greater than or equal to ' minDateUtc '.	The value of the parameter maxDateUtc is lesser than the value of the parameter minDateUtc .
TranslateApiException	IP is over the quota.	<ul style="list-style-type: none"> 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.
TranslateApiException	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.

View code examples on GitHub

- [C#](#)
- [PHP](#)

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

```
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
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 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()
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 ' maxDateUtc ' must be greater than or equal to ' minDateUtc '.	The value of the parameter maxDateUtc is lesser than the value of the parameter minDateUtc .

EXCEPTION	MESSAGE	CONDITIONS
TranslateApiException	IP is over the quota.	<ul style="list-style-type: none"> • 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.
TranslateApiException	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.

View code examples on GitHub

- [C#](#)
- [PHP](#)

How to return N-Best translations

11/8/2019 • 2 minutes to read • [Edit Online](#)

NOTE

This method is deprecated. It is not available in V3.0 of the Translator Text API.

The `GetTranslations()` and `GetTranslationsArray()` methods of the Microsoft Translator API include an optional Boolean flag "IncludeMultipleMTAlternatives". The method will return up to `maxTranslations` alternatives where the delta is supplied from the N-Best list of the translator engine.

The signature is:

Syntax

C#

```
GetTranslationsResponse Microsoft.Translator.GetTranslations(appId, text, from, to, maxTranslations, options);
```

Parameters

PARAMETER	DESCRIPTION
appId	Required If the Authorization header is used, leave the appId field empty else specify a string containing "Bearer" + " " + access token.
text	Required A string representing the text to translate. The size of the text must not exceed 10000 characters.
from	Required A string representing the language code of the text to translate.
to	Required A string representing the language code to translate the text into.
maxTranslations	Required An int representing the maximum number of translations to return.
options	Optional A TranslateOptions object that contains the values listed below. They are all optional and default to the most common settings.

- Category: The only supported, and the default, option is "general".
- ContentType: The only supported, and the default, option is "text/plain".
- State: User state to help correlate request and response. The same contents will be returned in the response.
- IncludeMultipleMTAlternatives: flag to determine whether to return more than one alternatives from the MT engine. Default is false and includes only 1 alternative.

Ratings

The ratings are applied as follows: The best automatic translation has a rating of 5. The automatically generated (N-Best) translation alternatives have a rating of 0, and have a match degree of 100.

Number of alternatives

The number of returned alternatives is up to maxTranslations, but may be less.

Language pairs

This functionality is not available for translations between Simplified and Traditional Chinese, both directions. It is available for all other Microsoft Translator supported language pairs.

Language and region support for the Translator Text API

12/2/2019 • 5 minutes to read • [Edit Online](#)

The Translator Text API supports the following languages for text to text translation. Neural Machine Translation (NMT) is the new standard for high-quality AI-powered machine translations and is available as the default using V3 of the Translator Text API when a neural system is available.

[Learn more about how machine translation works](#)

Translation

V2 Translator API

NOTE

V2 was deprecated on April 30, 2018. Please migrate your applications to V3 in order to take advantage of new functionality available exclusively in V3.

- Statistical only: No neural system is available for this language.
- Neural available: A neural system is available. Use the parameter `category=generalnn` to access the neural system.
- Neural default: Neural is the default translation system. Use the parameter `category=smt` to access the statistical system for use with the Microsoft Translator Hub.
- Neural only: Only neural translation is available.

V3 Translator API The V3 Translator API is neural by default and statistical systems are only available when no neural system exists.

NOTE

Currently, a subset of the neural languages are available in Custom Translator and we are gradually adding additional ones. [View languages currently available in Custom Translator.](#)

LANGUAGE	LANGUAGE CODE	V2 API	V3 API
Afrikaans	<code>af</code>	Statistical only	Neural
Arabic	<code>ar</code>	Neural available	Neural
Bangla	<code>bn</code>	Neural available	Neural
Bosnian (Latin)	<code>bs</code>	Neural available	Neural
Bulgarian	<code>bg</code>	Neural available	Neural
Cantonese (Traditional)	<code>yue</code>	Statistical only	Statistical

LANGUAGE	LANGUAGE CODE	V2 API	V3 API
Catalan	ca	Statistical only	Statistical
Chinese Simplified	zh-Hans	Neural default	Neural
Chinese Traditional	zh-Hant	Neural default	Neural
Croatian	hr	Neural available	Neural
Czech	cs	Neural available	Neural
Danish	da	Neural available	Neural
Dutch	nl	Neural available	Neural
English	en	Neural available	Neural
Estonian	et	Neural available	Neural
Fijian	fj	Statistical only	Statistical
Filipino	fil	Statistical only	Statistical
Finnish	fi	Neural available	Neural
French	fr	Neural available	Neural
German	de	Neural available	Neural
Greek	el	Neural available	Neural
Haitian Creole	ht	Statistical only	Statistical
Hebrew	he	Neural available	Neural
Hindi	hi	Neural default	Neural
Hmong Daw	mww	Statistical only	Statistical
Hungarian	hu	Neural available	Neural
Icelandic	is	Neural only	Neural
Indonesian	id	Statistical only	Statistical
Italian	it	Neural available	Neural
Japanese	ja	Neural available	Neural

LANGUAGE	LANGUAGE CODE	V2 API	V3 API
Kiswahili	sw	Statistical only	Statistical
Klingon	tlh	Statistical only	Statistical
Klingon (plqaD)	tlh-Qaak	Statistical only	Statistical
Korean	ko	Neural available	Neural
Latvian	lv	Neural available	Neural
Lithuanian	lt	Neural available	Neural
Malagasy	mg	Statistical only	Statistical
Malay	ms	Statistical only	Statistical
Maltese	mt	Statistical only	Statistical
Maori	mi	Neural only	Neural
Norwegian	nb	Neural available	Neural
Persian	fa	Neural available	Neural
Polish	pl	Neural available	Neural
Portuguese	pt	Neural available	Neural
Queretaro Otomi	otq	Statistical only	Statistical
Romanian	ro	Neural available	Neural
Russian	ru	Neural available	Neural
Samoan	sm	Statistical only	Statistical
Serbian (Cyrillic)	sr-Cyrl	Statistical only	Statistical
Serbian (Latin)	sr-Latn	Statistical only	Statistical
Slovak	sk	Neural available	Neural
Slovenian	sl	Neural available	Neural
Spanish	es	Neural available	Neural
Swedish	sv	Neural available	Neural

LANGUAGE	LANGUAGE CODE	V2 API	V3 API
Tahitian	ty	Statistical only	Statistical
Tamil	ta	Neural available	Neural
Telugu	te	Neural only	Neural
Thai	th	Neural available	Neural
Tongan	to	Statistical only	Statistical
Turkish	tr	Neural available	Neural
Ukrainian	uk	Neural available	Neural
Urdu	ur	Statistical only	Statistical
Vietnamese	vi	Neural available	Neural
Welsh	cy	Neural available	Neural
Yucatec Maya	yua	Statistical only	Statistical

Transliteration

The Transliterate method supports the following languages. In the "To/From", "<-->" indicates that the language can be transliterated from or to either of the scripts listed. The "-->" indicates that the language can only be transliterated from one script to the other.

LANGUAGE	LANGUAGE CODE	SCRIPT	TO/FROM	SCRIPT
Arabic	ar	Arabic Arab	<-->	Latin Latn
Bangla	bn	Bengali Beng	<-->	Latin Latn
Chinese (Simplified)	zh-Hans	Chinese Simplified Hans	<-->	Latin Latn
Chinese (Simplified)	zh-Hans	Chinese Simplified Hans	<-->	Chinese Traditional Hant
Chinese (Traditional)	zh-Hant	Chinese Traditional Hant	<-->	Latin Latn
Chinese (Traditional)	zh-Hant	Chinese Traditional Hant	<-->	Chinese Simplified Hans
Gujarati	gu	Gujarati Gujr	-->	Latin Latn
Hebrew	he	Hebrew Hebr	<-->	Latin Latn

LANGUAGE	LANGUAGE CODE	SCRIPT	TO/FROM	SCRIPT
Hindi	hi	Devanagari Deva	<-->	Latin Latn
Japanese	ja	Japanese Jpan	<-->	Latin Latn
Kannada	kn	Kannada Knda	-->	Latin Latn
Malayalam	ml	Malayalam Mlym	-->	Latin Latn
Marathi	mr	Devanagari Deva	-->	Latin Latn
Oriya	or	Oriya Orya	<-->	Latin Latn
Punjabi	pa	Gurmukhi Guru	<-->	Latin Latn
Serbian (Cyrillic)	sr-Cyr1	Cyrillic Cyr1	-->	Latin Latn
Serbian (Latin)	sr-Latn	Latin Latn	-->	Cyrillic Cyr1
Tamil	ta	Tamil Tam1	-->	Latin Latn
Telugu	te	Telugu Telu	-->	Latin Latn
Thai	th	Thai Thai	<-->	Latin Latn

Dictionary

The dictionary supports the following languages to or from English using the Lookup and Examples methods.

LANGUAGE	LANGUAGE CODE
Afrikaans	af
Arabic	ar
Bangla	bn
Bosnian (Latin)	bs
Bulgarian	bg
Catalan	ca
Chinese Simplified	zh-Hans
Croatian	hr
Czech	cs

LANGUAGE	LANGUAGE CODE
Danish	da
Dutch	nl
Estonian	et
Finnish	fi
French	fr
German	de
Greek	el
Haitian Creole	ht
Hebrew	he
Hindi	hi
Hmong Daw	mww
Hungarian	hu
Icelandic	is
Indonesian	id
Italian	it
Japanese	ja
Kiswahili	sw
Klingon	tlh
Korean	ko
Latvian	lv
Lithuanian	lt
Malay	ms
Maltese	mt
Norwegian	nb

LANGUAGE	LANGUAGE CODE
Persian	fa
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Serbian (Latin)	sr-Latn
Slovak	sk
Slovenian	sl
Spanish	es
Swedish	sv
Tamil	ta
Thai	th
Turkish	tr
Ukrainian	uk
Urdu	ur
Vietnamese	vi
Welsh	cy

Detect

Translator Text API detects all languages available for translation and transliteration.

Access the Translator Text API language list programmatically

You can retrieve a list of supported languages for the Translator Text API v3.0 using the `Languages` method. You can view the list by feature, language code, as well as the language name in English or any other supported language. This list is automatically updated by the Microsoft Translator service as new languages are made available.

[View Languages operation reference documentation](#)

Customization

The following languages are available for customization to or from English using [Custom Translator](#).

LANGUAGE	LANGUAGE CODE
Arabic	ar
Bangla	bn
Bosnian (Latin)	bs
Bulgarian	bg
Chinese Simplified	zh-Hans
Chinese Traditional	zh-Hant
Croatian	hr
Czech	cs
Danish	da
Dutch	nl
English	en
Estonian	et
Finnish	fi
French	fr
German	de
Greek	el
Hebrew	he
Hindi	hi
Hungarian	hu
Icelandic	is
Indonesian	id
Irish	ga
Italian	it
Japanese	ja

LANGUAGE	LANGUAGE CODE
Kiswahili	sw
Korean	ko
Latvian	lv
Lithuanian	lt
Malagasy	mg
Maori	mi
Norwegian	nb
Persian	fa
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Samoan	sm
Serbian (Latin)	sr-Latn
Slovak	sk
Slovenian	sl
Spanish	es
Swedish	sv
Thai	th
Turkish	tr
Ukrainian	uk
Vietnamese	vi
Welsh	cy

Access the list on the [Microsoft Translator website](#)

For a quick look at the languages, the Microsoft Translator website shows all the languages supported by the Translator Text and Speech APIs. This list doesn't include developer-specific information such as language codes.

[See the list of languages](#)

Request limits for Translator Text

11/26/2019 • 2 minutes to read • [Edit Online](#)

This article provides throttling limits for the Translator Text API. Services include translation, transliteration, sentence length detection, language detection, and alternate translations.

Character and array limits per request

Each Translate request is limited to 5,000 characters. You're charged per character, not by the number of requests. It's recommended to send shorter requests.

The following table lists array element and character limits for each operation of the Translator Text API.

OPERATION	MAXIMUM SIZE OF ARRAY ELEMENT	MAXIMUM NUMBER OF ARRAY ELEMENTS	MAXIMUM REQUEST SIZE (CHARACTERS)
Translate	5,000	100	5,000
Transliterate	5,000	10	5,000
Detect	10,000	100	50,000
BreakSentence	10,000	100	50,000
Dictionary Lookup	100	10	1,000
Dictionary Examples	100 for text and 100 for translation (200 total)	10	2,000

Character limits per hour

Your character limit per hour is based on your Translator Text subscription tier.

The hourly quota should be consumed evenly throughout the hour. For example, at the F0 tier limit of 2 million characters per hour, characters should be consumed no faster than roughly 33,300 characters per minute sliding window (2 million characters divided by 60 minutes).

If you reach or surpass these limits, or send too large of a portion of the quota in a short period of time, you'll likely receive an out of quota response. There are no limits on concurrent requests.

TIER	CHARACTER LIMIT
F0	2 million characters per hour
S1	40 million characters per hour
S2 / C2	40 million characters per hour
S3 / C3	120 million characters per hour

TIER	CHARACTER LIMIT
S4 / C4	200 million characters per hour

Limits for [multi-service subscriptions](#) are the same as the S1 tier.

These limits are restricted to Microsoft's standard translation models. Custom translation models that use Custom Translator are limited to 1,800 characters per second.

Latency

The Translator Text API has a maximum latency of 15 seconds using standard models and 120 seconds when using custom models. Typically, responses *for text within 100 characters* are returned in 150 milliseconds to 300 milliseconds. The custom translator models have similar latency characteristics on sustained request rate and may have a higher latency when your request rate is intermittent. Response times will vary based on the size of the request and language pair. If you don't receive a translation or an [error response](#) within that timeframe, please check your code, your network connection and retry.

Sentence length limits

When using the [BreakSentence](#) function, sentence length is limited to 275 characters. There are exceptions for these languages:

LANGUAGE	CODE	CHARACTER LIMIT
Chinese	zh	132
German	de	290
Italian	it	280
Japanese	ja	150
Portuguese	pt	290
Spanish	es	280
Italian	it	280
Thai	th	258

NOTE

This limit doesn't apply to translations.

Next steps

- [Pricing](#)
- [Regional availability](#)
- [v3 Translator Text API reference](#)

What is Custom Translator?

12/10/2019 • 3 minutes to read • [Edit Online](#)

[Custom Translator](#) is a feature of the Microsoft Translator service, which enables Translator enterprises, app developers, and language service providers to build customized neural machine translation (NMT) systems. The customized translation systems seamlessly integrate into existing applications, workflows, and websites. [Custom Translator](#) offers similar capabilities to what [Microsoft Translator Hub](#) does for Statistical Machine Translation (SMT), but exclusively for Neural Machine Translation (NMT) systems.

Translation systems built with [Custom Translator](#) are available through the same cloud-based, [secure](#), high performance, highly scalable Microsoft Translator [Text API V3](#), that powers billions of translations every day.

Custom Translator supports more than three dozen languages, and maps directly to the languages available for NMT. For a complete list, see [Microsoft Translator Languages](#).

Features

Custom Translator provides different features to build custom translation system and subsequently access it.

FEATURE	DESCRIPTION
Leverage neural machine translation technology	Improve your translation by leveraging neural machine translation (NMT) provided by Custom translator.
Build systems that knows your business terminology	Customize and build translation systems using parallel documents, that understand the terminologies used in your own business and industry.
Use a dictionary to build your models	If you don't have training data set, you can train a model with only dictionary data.
Collaborate with others	Collaborate with your team by sharing your work with different people.
Access your custom translation model	Your custom translation model can be accessed anytime by your existing applications/ programs via Microsoft Translator Text API V3.

Get better translations

Microsoft Translator released [Neural Machine Translation \(NMT\)](#) in 2016. NMT provided major advances in translation quality over the industry-standard [Statistical Machine Translation \(SMT\)](#) technology. Because NMT better captures the context of full sentences before translating them, it provides higher quality, more human-sounding, and more fluent translations. [Custom Translator](#) provides NMT for your custom models resulting better translation quality.

You can use previously translated documents to build a translation system. These documents include domain-specific terminology and style, better than a standard translation system. Users can upload ALIGN, PDF, LCL, HTML, HTM, XLF, TMX, XLIFF, TXT, DOCX, and XLSX documents.

Custom Translator also accepts data that's parallel at the document level to make data collection and preparation more effective. If users have access to versions of the same content in multiple languages but in separate

documents, Custom Translator will be able to automatically match sentences across documents.

If the appropriate type and amount of training data is supplied, it's not uncommon to see [BLEU score](#) gains between 5 and 10 points by using Custom Translator.

Be productive and cost effective

With [Custom Translator](#), training and deploying a custom system doesn't require any programming skills.

Using the secure [Custom Translator](#) portal, users can upload training data, train systems, test systems, and deploy them to a production environment through an intuitive user interface. The system will then be available for use at scale within a few hours (actual time depends on training data size).

[Custom Translator](#) can also be programmatically accessed through a [dedicated API](#) (currently in preview). The API allows users to manage creating or updating training on a regular basis through their own app or webservice.

The cost of using a custom model to translate content is based on the user's Translator Text API pricing tier. See the Cognitive Services [Translator Text API pricing webpage](#) for pricing tier details.

Securely translate anytime, anywhere on all your apps and services

Custom systems can be seamlessly accessed and integrated into any product or business workflow, and on any device, via the Microsoft Translator Text API through standard REST technology.

Next steps

- Read about [pricing details](#).
- With [Quickstart](#) learn to build a translation model in Custom Translator.

Quickstart: Build, deploy, and use a custom model for translation

12/10/2019 • 2 minutes to read • [Edit Online](#)

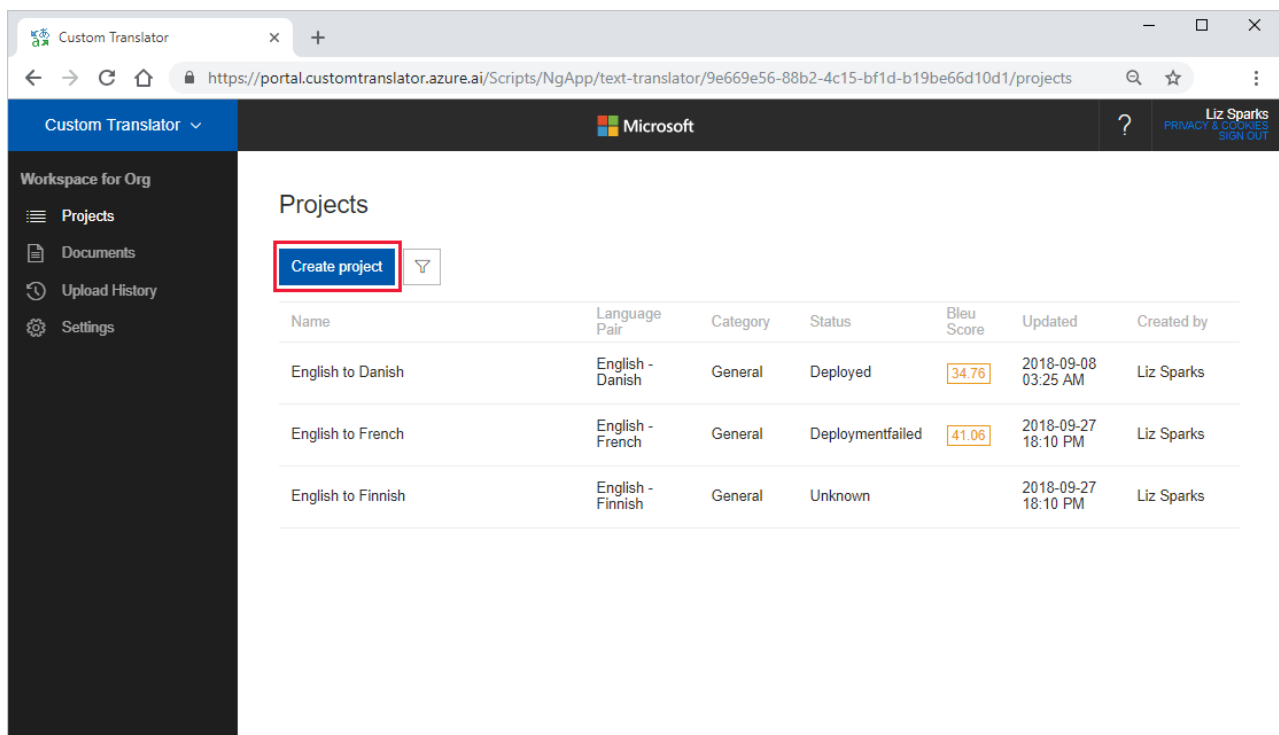
This article provides step-by-step instructions to build a translation system with Custom Translator.

Prerequisites

1. To use the [Custom Translator](#) Portal, you will need a [Microsoft account](#) or [Azure AD account](#) (organization account hosted on Azure) to sign in.
2. A subscription to the Translator Text API via the Azure portal. You will need the Translator Text API subscription key to associate with your workspace in Custom Translator. See [how to sign up for the Translator Text API](#).
3. When you have both of the above, sign in to the [Custom Translator](#) portal. Once on the Custom Translator portal, navigate to the Settings page where you can associate your Microsoft Translator Text API subscription key with your workspace.

Create a project

On the Custom Translator portal landing page, click New Project. On the dialog you can enter your desired project name, language pair, and category, as well as other relevant fields. Then, save your project. For more details, visit [Create Project](#).



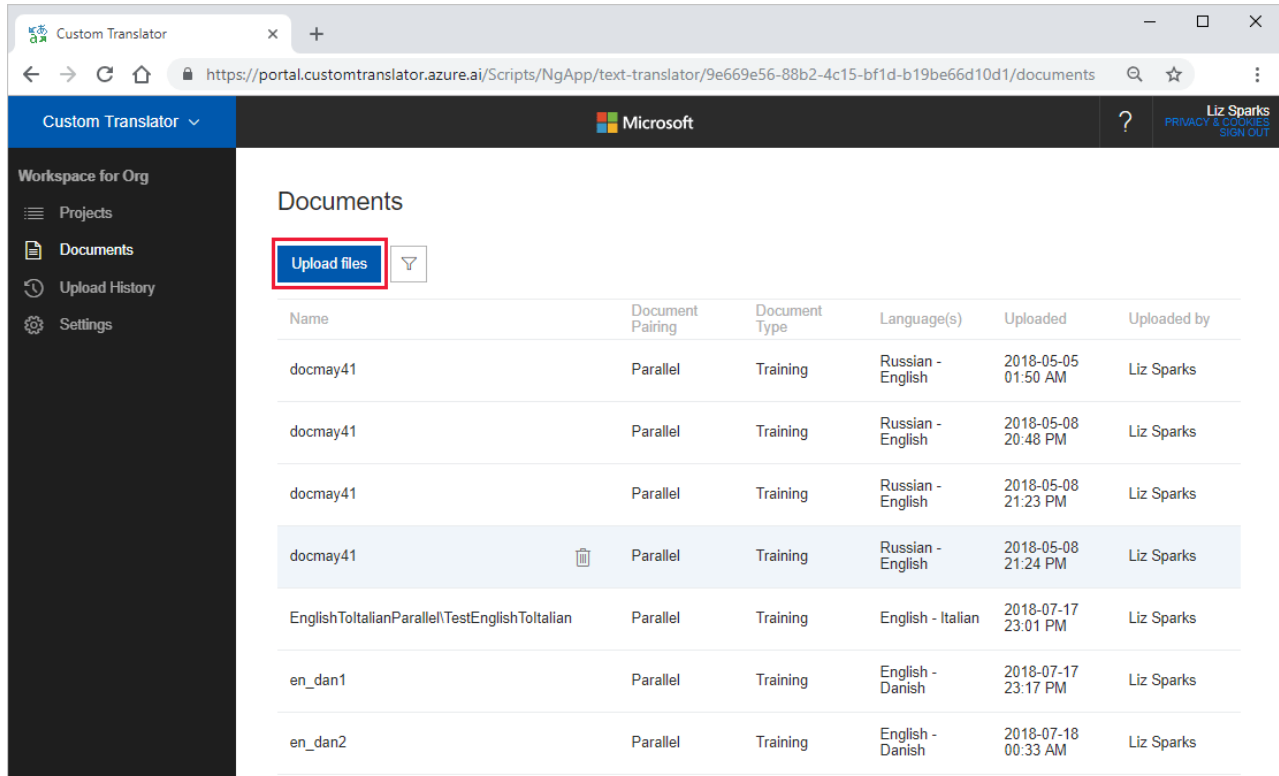
The screenshot shows the Custom Translator portal interface. On the left is a sidebar with navigation options: Projects, Documents, Upload History, and Settings. The main area is titled 'Projects' and features a 'Create project' button (highlighted with a red box) and a filter icon. Below this is a table listing existing projects.

Name	Language Pair	Category	Status	Bleu Score	Updated	Created by
English to Danish	English - Danish	General	Deployed	34.76	2018-09-08 03:25 AM	Liz Sparks
English to French	English - French	General	Deploymentfailed	41.06	2018-09-27 18:10 PM	Liz Sparks
English to Finnish	English - Finnish	General	Unknown		2018-09-27 18:10 PM	Liz Sparks


Upload documents

Next, upload [training](#), [tuning](#) and [testing](#) document sets. You can upload both [parallel](#) and [combo](#) documents. You can also upload [dictionary](#).

You can upload documents from either the documents tab or from a specific project's page.



The screenshot shows the 'Custom Translator' interface. On the left is a sidebar with 'Workspace for Org' and navigation links: 'Projects', 'Documents' (selected), 'Upload History', and 'Settings'. The main area is titled 'Documents' and features an 'Upload files' button (highlighted with a red box) and a search icon. Below this is a table of uploaded documents.

Name	Document Pairing	Document Type	Language(s)	Uploaded	Uploaded by
docmay41	Parallel	Training	Russian - English	2018-05-05 01:50 AM	Liz Sparks
docmay41	Parallel	Training	Russian - English	2018-05-08 20:48 PM	Liz Sparks
docmay41	Parallel	Training	Russian - English	2018-05-08 21:23 PM	Liz Sparks
docmay41	 Parallel	Training	Russian - English	2018-05-08 21:24 PM	Liz Sparks
EnglishToItalianParallelTestEnglishToItalian	Parallel	Training	English - Italian	2018-07-17 23:01 PM	Liz Sparks
en_dan1	Parallel	Training	English - Danish	2018-07-17 23:17 PM	Liz Sparks
en_dan2	Parallel	Training	English - Danish	2018-07-18 00:33 AM	Liz Sparks

When uploading documents, choose the document type (training, tuning, or testing), and the language pair. When uploading parallel documents, you'll need to additionally specify a document name. For more details, visit [Upload document](#).

Create a model

When all your required documents are uploaded the next step is to build your model.

Select the project you've created. You'll see all the documents you've uploaded that share a language pair with this project. Select the documents that you want included in your model. You can select [training](#), [tuning](#), and [testing](#) data or select just training data and let Custom Translator automatically build tuning and test sets for your model.

Custom Translator

Microsoft

Workspace for Org

- Projects
- Documents
- Upload History
- Settings

Projects > English to Danish

Category ID: 00000000-0000-0000-0000-000000000000-GENERAL
 Language Pair: English - Danish
 Category: General
[Edit project](#)

Data Models

Create model Upload files

6 Documents

<input type="checkbox"/>	Name	Document Pairing	Document Type	Language(s)	English Sentences	Danish Sentences
<input type="checkbox"/>	en_dan1	Parallel	Training	English - Danish	60,109	60,109
<input type="checkbox"/>	en_dan2	Parallel	Training	English - Danish	86,980	86,980
<input type="checkbox"/>	en_dan3	Parallel	Training	English - Danish	79,487	79,487
<input type="checkbox"/>	English to Danish Tuning 2	Parallel	Tuning	English - Danish	3,500	3,500
<input type="checkbox"/>	EU_15900000	Parallel	Tuning	Danish - English	4,355	4,355
<input type="checkbox"/>	English to Danish Testing 2 - Para	Parallel	Testing	English - Danish	3,050	3,050

When you've finished selecting your desired documents, click Create Model button to create your model and start training. You can see the status of your training, and details for all the models you've trained, in the Models tab.

For more details, visit [Create a Model](#).

Analyze your model

Once your training has completed successfully, inspect the results. The BLEU score is one metric that indicates the quality of your translation. You can also manually compare the translations made with your custom model to the translations provided in your test set by navigating to the "Test" tab and clicking "System Results." Manually inspecting a few of these translations will give you a good idea of the quality of translation produced by your system. For more details, visit [System Test Results](#).

Deploy a trained model

When you are ready to deploy your trained model, click the "Deploy" button. You can have one deployed model per project, and you can view the status of your deployment in the Status column. For more details, visit [Model Deployment](#)

Custom Translator

Microsoft

Workspace for Org

- Projects
- Documents
- Upload History
- Settings

Projects > English to Danish

Category ID: 00000000-0000-0000-0000-000000000000-GENERAL
Language Pair: English - Danish
Category: General
[Edit project](#)

Data Models

4 models

Name	Status	Bleu Score	Baseline Bleu Score	Training	Dictionary	Tuning	Test	Model Action
English >> Danish 1st model	Undeployed	34.76	34.27	52,015	0	2,500	2,500	Deploy
English >> Danish 2nd model	Succeeded	34.75	34.52	52,015	0	2,500	2,500	Deploy
English >> Danish 3rd model	Succeeded	31	30.88	57,015	0	4,139	2,868	Deploy
Second model for EN to DAN	Succeeded	34.58	34.4	52,015	0	2,500	2,500	Deploy

Use a deployed model

Deployed models can be accessed via the Microsoft Translator [Text API V3](#) by specifying the [CategoryID](#). More information about the Translator Text API can be found on the [API Reference](#) webpage.

Next steps

- Learn how to navigate the [Custom Translator workspace and manage your projects](#).

What is a Custom Translator workspace?

11/5/2019 • 3 minutes to read • [Edit Online](#)

A workspace is a work area for composing and building your custom translation system. A workspace can contain multiple projects, models, and documents. All the work you do in Custom Translator is inside a specific workspace.

Workspace is private to you and the people you invite into your workspace. Uninvited people do not have access to the content of your workspace. You can invite as many people as you like into your workspace and modify or remove their access anytime. You can also create a new workspace. By default a workspace will not contain any projects or documents that are within your other workspaces.

What is a Custom Translator project?

A project is a wrapper for a model, documents, and tests. Each project automatically includes all documents that are uploaded into that workspace that have the correct language pair. For example, if you have both an English to Spanish project and a Spanish to English project, the same documents will be included in both projects. Each project has a CategoryID associated with it that is used when querying the [V3 API](#) for translations. CategoryID is parameter used to get translations from a customized system built with Custom Translator.

Project categories

The category identifies the domain – the area of terminology and style you want to use – for your project. Choose the category most relevant to your documents. In some cases, your choice of the category directly influences the behavior of the Custom Translator.

We have two sets of baseline models. They are General and Technology. If the category **Technology** is selected, the Technology baseline models will be used. For any other category selection, the General baseline models are used. The Technology baseline model does well in technology domain, but it shows lower quality, if the sentences used for translation don't fall within the technology domain. We suggest customers to select category Technology only if sentences fall strictly within the technology domain.

In the same workspace, you may create projects for the same language pair in different categories. Custom Translator prevents creation of a duplicate project with the same language pair and category. Applying a label to your project allows you to avoid this restriction. Don't use labels unless you're building translation systems for multiple clients, as adding a unique label to your project will be reflected in your projects CategoryID.

Project labels

Custom Translator allows you to assign a project label to your project. The project label distinguishes between multiple projects with the same language pair and category. As a best practice, avoid using project labels unless necessary.

The project label is used as part of the CategoryID. If the project label is left unset or is set identically across projects, then projects with the same category and *different* language pairs will share the same CategoryID. This approach is advantageous because it allows you or your customer to switch between languages when using the Text Translator API without worrying about a CategoryID that is unique to each project.

For example, if I wanted to enable translations in the Technology domain from English to French and from French to English, I would create two projects: one for English -> French, and one for French -> English. I would specify the same category (Technology) for both and leave the project label blank. The CategoryID for both projects would match, so I could query the API for both English and French translations without having to modify my CategoryID.

If you are a language service provider and want to serve multiple customers with different models that retain the same category and language pair, then using a project label to differentiate between customers would be a wise decision.

Next steps

- Read about [Training and model](#) to know, how to efficiently build a translation model.

What are parallel documents?

10/20/2019 • 2 minutes to read • [Edit Online](#)

Parallel documents are pairs of documents where one is the translation of the other. One document in the pair contains sentences in the source language and the other document contains these sentences translated into the target language. It doesn't matter which language is marked as "source" and which language is marked as "target" – a parallel document can be used to train a translation system in either direction.

Requirements

You will need a minimum of 10,000 unique aligned parallel sentences to train a system. This limitation is a safety net to ensure your parallel sentences contain enough unique vocabulary to successfully train a translation model. As a best practice, continuously add more parallel content and retrain to improve the quality of your translation system. Please refer to [Sentence Alignment](#).

Microsoft requires that documents uploaded to the Custom Translator do not violate a third party's copyright or intellectual properties. For more information, please see the [Terms of Use](#). Uploading a document using the portal does not alter the ownership of the intellectual property in the document itself.

Use of parallel documents

Parallel documents are used by the system:

1. To learn how words, phrases and sentences are commonly mapped between the two languages.
2. To learn how to process the appropriate context depending on the surrounding phrases. A word may not always translate to the exact same word in the other language.

As a best practice, make sure that there is a 1:1 sentence correspondence between the source and target language versions of the documents.

If your project is domain (category) specific, your documents should be consistent in terminology within that category. The quality of the resulting translation system depends on the number of sentences in your document set and the quality of the sentences. The more examples your documents contain with diverse usages for a word specific to your category, the better job the system can do during translation.

Documents uploaded are private to each workspace and can be used in as many projects or trainings as you like. Sentences extracted from your documents are stored separately in your repository as plain Unicode text files and are available for you delete. Do not use the Custom Translator as a document repository, you will not be able to download the documents you uploaded in the format you uploaded them.

Next steps

- Learn how to use a [dictionary](#) in Custom Translator.

What is a dictionary?

10/4/2019 • 3 minutes to read • [Edit Online](#)

A dictionary is an aligned pair of documents that specifies a list of phrases or sentences and their corresponding translations. Use a dictionary in your training, when you want Microsoft Translator to always translate any instances of the source phrase or sentence, using the translation you've provided in the dictionary. Dictionaries are sometimes called glossaries or term bases. You can think of the dictionary as a brute force "copy and replace" for all the terms you list. Furthermore, Microsoft Custom Translator service builds and makes use of its own general purpose dictionaries to improve the quality of its translation. However, a customer provided dictionary takes precedent and will be searched first to lookup words or sentences.

Dictionaries only work for projects in language pairs that have a fully supported Microsoft general neural network model behind them. [View the complete list of languages.](#)

Phrase dictionary

When you include a phrase dictionary in training your model, any word or phrase listed is translated in the way you specified. The rest of the sentence is translated as usual. You can use a phrase dictionary to specify phrases that shouldn't be translated by providing the same untranslated phrase in the source and target file in the dictionary.

Sentence dictionary

The sentence dictionary allows you to specify an exact target translation for a source sentence. For a sentence dictionary match to occur, the entire submitted sentence must match the source dictionary entry. If only a portion of the sentence matches, the entry won't match. When a match is detected, the target entry of the sentence dictionary will be returned.

Dictionary-only trainings

You can train a model using only dictionary data. To do this, select only the dictionary document (or multiple dictionary documents) that you wish to include and tap Create model. Since this is a dictionary-only training, there is no minimum number of training sentences required. Your model will typically complete training much faster than a standard training. The resulting models will use the Microsoft baseline models for translation with the addition of the dictionaries you have added. You will not get a test report.

NOTE

Custom Translator does not sentence align dictionary files, so it is important that there are an equal number of source and target phrases/sentences in your dictionary documents and that they are precisely aligned.

Recommendations

- Dictionaries are not a substitute for training a model using training data. It is recommended to avoid them and let the system learn from your training data. However, when sentences or compound nouns must be rendered as-is, use a dictionary.
- The phrase dictionary should be used sparingly. So, be aware that when a phrase within a sentence is replaced, the context within that sentence is lost or limited for translating the rest of the sentence. The result is that while the phrase or word within the sentence will translate according to the provided dictionary, the overall

translation quality of the sentence will often suffer.

- The phrase dictionary works well for compound nouns like product names ("Microsoft SQL Server"), proper names ("City of Hamburg"), or features of the product ("pivot table"). It does not work equally well for verbs or adjectives because these are typically highly inflected in the source or in the target language. Best practices is to avoid phrase dictionary entries for anything but compound nouns.
- When using a phrase dictionary, capitalization and punctuation are important. Dictionary entries will only match words and phrases in the input sentence that use exactly the same capitalization and punctuation as specified in the source dictionary file. Also the translations will reflect the capitalization and punctuation provided in the target dictionary file. For example, if you trained an English to Spanish system that uses a phrase dictionary that specifies "US" in the source file, and "EE.UU." in the target file. When you request translation of a sentence that includes the word "us" (not capitalized), this would NOT match the dictionary. However if you request translation of a sentence that contains the word "US" (capitalized) then it would match the dictionary and the translation would contain "EE.UU." Note that the capitalization and punctuation in the translation may be different than specified in the dictionary target file, and may be different from the capitalization and punctuation in the source. It follows the rules of the target language.
- If a word appears more than once in a dictionary file, the system will always use the last entry provided. Hence, your dictionary should not contain multiple translations of the same word.

Next steps

- Read about [guidelines on document formats](#).

Document formats and naming convention guidance

7/28/2019 • 2 minutes to read • [Edit Online](#)

Any file used for custom translation must be at least **four** characters in length.

This table includes all supported file formats that you can use to build your translation system:

FORMAT	EXTENSIONS	DESCRIPTION
XLIFF	.XLF, .XLIFF	A parallel document format, export of Translation Memory systems. The languages used are defined inside the file.
TMX	.TMX	A parallel document format, export of Translation Memory systems. The languages used are defined inside the file.
ZIP	.ZIP	ZIP is an archive file format.
Locstudio	.LCL	A Microsoft format for parallel documents
Microsoft Word	.DOCX	Microsoft Word document
Adobe Acrobat	.PDF	Adobe Acrobat portable document
HTML	.HTML, .HTM	HTML document
Text file	.TXT	UTF-16 or UTF-8 encoded text files. The file name must not contain Japanese characters.
Aligned text file	.ALIGN	The extension <code>.ALIGN</code> is a special extension that you can use if you know that the sentences in the document pair are perfectly aligned. If you provide a <code>.ALIGN</code> file, Custom Translator will not align the sentences for you.
Excel file	.XLSX	Excel file (2013 or later). First line/ row of the spreadsheet should be language code.

Dictionary formats

For dictionaries, Custom Translator supports all file formats that are supported for training sets. If you are using an Excel dictionary, the first line/ row of the spreadsheet should be language codes.

Zip file formats

Documents can be grouped into a single zip file and uploaded. The Custom Translator supports zip file formats (ZIP, GZ, and TGZ).

Each document in the zip file with the extension TXT, HTML, HTM, PDF, DOCX, ALIGN must follow this naming convention:

{document name}_{language code} where {document name} is the name of your document, {language code} is the ISO LanguageID (two characters), indicating that the document contains sentences in that language. There must be an underscore (_) before the language code.

For example, to upload two parallel documents within a zip for an English to Spanish system, the files should be named "data_en" and "data_es".

Translation Memory files (TMX, XLF, XLIFF, LCL, XLSX) are not required to follow the specific language naming convention.

Next steps

- Read about the [project](#) to create and manage them.

Sentence pairing and alignment in parallel documents

10/20/2019 • 2 minutes to read • [Edit Online](#)

During the training, sentences present in parallel documents are paired or aligned. Custom Translator reports the number of sentences it was able to pair as the Aligned Sentences in each of the data sets.

Pairing and alignment process

Custom Translator learns translations of sentences one sentence at a time. It reads a sentence from source, and then the translation of this sentence from target. Then it aligns words and phrases in these two sentences to each other. This process enables it to create a map of the words and phrases in one sentence to the equivalent words and phrases in the translation of this sentence. Alignment tries to ensure that the system trains on sentences that are translations of each other.

Pre-aligned documents

If you know you have parallel documents, you may override the sentence alignment by supplying pre-aligned text files. You can extract all sentences from both documents into text file, organized one sentence per line, and upload with an `.align` extension. The `.align` extension signals Custom Translator that it should skip sentence alignment.

For best results, try to make sure that you have one sentence per line in your files. Don't have newline characters within a sentence as this will cause poor alignments.

Suggested minimum number of sentences

For a training to succeed, the table below shows the minimum number of sentences required in each document type. This limitation is a safety net to ensure your parallel sentences contain enough unique vocabulary to successfully train a translation model. The general guideline is having more in-domain parallel sentences of human translation quality should produce higher quality models.

DOCUMENT TYPE	SUGGESTED MINIMUM SENTENCE COUNT	MAXIMUM SENTENCE COUNT
Training	10,000	No upper limit
Tuning	5,000	2,500
Testing	5,000	2,500
Dictionary	0	No upper limit

NOTE

- Training will not start and will fail if the 10,000 minimum sentence count for Training is not met.
- Tuning and Testing are optional. If you do not provide them, the system will remove an appropriate percentage from Training to use for validation and testing.
- You can train a model using only dictionary data. Please refer to [What is Dictionary](#).

Next steps

- Learn how to use a [dictionary](#) in Custom Translator.

Data filtering

7/28/2019 • 2 minutes to read • [Edit Online](#)

When you submit documents to be used for training a custom system, the documents undergo a series of processing and filtering steps to prepare for training. These steps are explained here. The knowledge of the filtering may help you understand the sentence count displayed in custom translator as well as the steps you may take yourself to prepare the documents for training with Custom Translator.

Sentence alignment

If your document isn't in XLIFF, TMX, or ALIGN format, Custom Translator aligns the sentences of your source and target documents to each other, sentence by sentence. Translator doesn't perform document alignment – it follows your naming of the documents to find the matching document of the other language. Within the document, Custom Translator tries to find the corresponding sentence in the other language. It uses document markup like embedded HTML tags to help with the alignment.

If you see a large discrepancy between the number of sentences in the source and target side documents, your document may not have been parallel in the first place, or for other reasons couldn't be aligned. The document pairs with a large difference (>10%) of sentences on each side warrant a second look to make sure they're indeed parallel. Custom Translator shows a warning next to the document if the sentence count differs suspiciously.

Deduplication

Custom Translator removes the sentences that are present in test and tuning documents from training data. The removal happens dynamically inside of the training run, not in the data processing step. Custom Translator reports the sentence count to you in the project overview before such removal.

Length filter

- Remove sentences with only one word on either side.
- Remove sentences with more than 100 words on either side. Chinese, Japanese, Korean are exempt.
- Remove sentences with fewer than 3 characters. Chinese, Japanese, Korean are exempt.
- Remove sentences with more than 2000 characters for Chinese, Japanese, Korean.
- Remove sentences with less than 1% alpha characters.
- Remove dictionary entries containing more than 50 words.

White space

- Replace any sequence of white-space characters including tabs and CR/LF sequences with a single space character.
- Remove leading or trailing space in the sentence

Sentence end punctuation

Replace multiple sentence end punctuation characters with a single instance.

Japanese character normalization

Convert full width letters and digits to half-width characters.

Unescaped XML tags

Filtering transforms unescaped tags into escaped tags:

- `<` becomes `<`
- `>` becomes `>`
- `&` becomes `&`

Invalid characters

Custom Translator removes sentences that contain Unicode character U+FFFD. The character U+FFFD indicates a failed encoding conversion.

Next steps

- [Train a model](#) in Custom Translator.

What are trainings and models?

12/10/2019 • 4 minutes to read • [Edit Online](#)

A model is the system, which provides translation for a specific language pair. The outcome of a successful training is a model. When training a model, three mutually exclusive document types are required: training, tuning, and testing. Dictionary document type can also be provided. Please refer to [Sentence alignment](#).

If only training data is provided when queuing a training, Custom Translator will automatically assemble tuning and testing data. It will use a random subset of sentences from your training documents, and exclude these sentences from the training data itself.

Training document type for Custom Translator

Documents included in training set are used by the Custom Translator as the basis for building your model. During training execution, sentences that are present in these documents are aligned (or paired). You can take liberties in composing your set of training documents. You can include documents that you believe are of tangential relevance in one model. Again exclude them in another to see the impact in [BLEU \(Bilingual Evaluation Understudy\) score](#). As long as you keep the tuning set and test set constant, feel free to experiment with the composition of the training set. This approach is an effective way to modify the quality of your translation system.

You can run multiple trainings within a project and compare the [BLEU scores](#) across all training runs. When you are running multiple trainings for comparison, ensure same tuning/ test data is specified each time. Also make sure to also inspect the results manually in the "[Testing](#)" tab.

Tuning document type for Custom Translator

Parallel documents included in this set are used by the Custom Translator to tune the translation system for optimal results.

The tuning data is used during training to adjust all parameters and weights of the translation system to the optimal values. Choose your tuning data carefully: the tuning data should be representative of the content of the documents you intend to translate in the future. The tuning data has a major influence on the quality of the translations produced. Tuning enables the translation system to provide translations that are closest to the samples you provide in the tuning data. You do not need more than 2500 sentences in your tuning data. For optimal translation quality, it is recommended to select the tuning set manually by choosing the most representative selection of sentences.

When creating your tuning set, choose sentences that are a meaningful and representative length of the future sentences that you expect to translate. You should also choose sentences that have words and phrases that you intend to translate in the approximate distribution that you expect in your future translations. In practice, a sentence length of 7 to 10 words will produce the best results, because these sentences contain enough context to show inflection and provide a phrase length that is significant, without being overly complex.

A good description of the type of sentences to use in the tuning set is prose: actual fluent sentences. Not table cells, not poems, not lists of things, not only punctuation, or numbers in a sentence - regular language.

If you manually select your tuning data, it should not have any of the same sentences as your training and testing data. The tuning data has a significant impact on the quality of the translations - choose the sentences carefully.

If you are not sure what to choose for your tuning data, just select the training data and let Custom Translator select the tuning data for you. When you let the Custom Translator choose the tuning data automatically, it will use a random subset of sentences from your bilingual training documents and exclude these sentences from the

training material itself.

Testing dataset for Custom Translator

Parallel documents included in the testing set are used to compute the BLEU (Bilingual Evaluation Understudy) score. This score indicates the quality of your translation system. This score actually tells you how closely the translations done by the translation system resulting from this training match the reference sentences in the test data set.

The BLEU score is a measurement of the delta between the automatic translation and the reference translation. Its value ranges from 0 to 100. A score of 0 indicates that not a single word of the reference appears in the translation. A score of 100 indicates that the automatic translation exactly matches the reference: the same word is in the exact same position. The score you receive is the BLEU score average for all sentences of the testing data.

The test data should include parallel documents where the target language sentences are the most desirable translations of the corresponding source language sentences in the the source-target pair. You may want to use the same criteria you used to compose the tuning data. However, the testing data has no influence over the quality of the translation system. It is used exclusively to generate the BLEU score for you.

You don't need more than 2,500 sentences as the testing data. When you let the system choose the testing set automatically, it will use a random subset of sentences from your bilingual training documents, and exclude these sentences from the training material itself.

You can view the custom translations of the testing set, and compare them to the translations provided in your testing set, by navigating to the test tab within a model.

What is a BLEU score?

11/5/2019 • 2 minutes to read • [Edit Online](#)

BLEU (Bilingual Evaluation Understudy) is a measurement of the differences between an automatic translation and one or more human-created reference translations of the same source sentence.

Scoring process

The BLEU algorithm compares consecutive phrases of the automatic translation with the consecutive phrases it finds in the reference translation, and counts the number of matches, in a weighted fashion. These matches are position independent. A higher match degree indicates a higher degree of similarity with the reference translation, and higher score. Intelligibility and grammatical correctness are not taken into account.

How BLEU works?

BLEU's strength is that it correlates well with human judgment by averaging out individual sentence judgment errors over a test corpus, rather than attempting to devise the exact human judgment for every sentence.

A more extensive discussion of BLEU scores is [here](#).

BLEU results depend strongly on the breadth of your domain, the consistency of the test data with the training and tuning data, and how much data you have available to train. If your models have been trained on a narrow domain, and your training data is consistent with your test data, you can expect a high BLEU score.

NOTE

A comparison between BLEU scores is only justifiable when BLEU results are compared with the same Test set, the same language pair, and the same MT engine. A BLEU score from a different test set is bound to be different.

Create a project

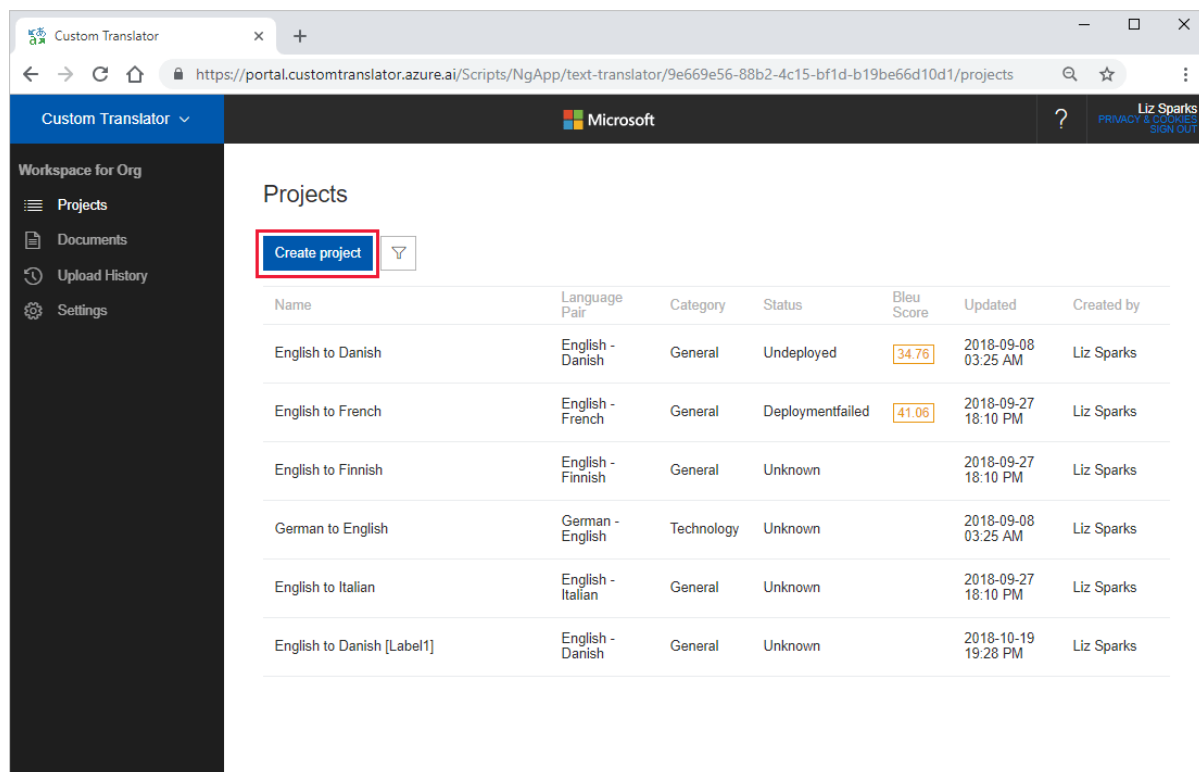
11/8/2019 • 2 minutes to read • [Edit Online](#)

A project is a container for a models, documents, and tests. Each project automatically includes all documents that are uploaded into that workspace that have the correct language pair.

Creating project is the first step toward building a model.

Create a project:

1. In the [Custom Translator](#) portal, click Create project.



2. Enter the following details about your project in the dialog:

- a. Project name (required): Give your project a unique, meaningful name. It's not necessary to mention the languages within the title.
- b. Description: A short summary about the project. This description has no influence over the behavior of the Custom Translator or your resulting custom system, but can help you differentiate between different projects.
- c. Language pair (required): Select the language that you're translating from and to.
- d. Category (required): Select the category that's most appropriate for your project. The category describes the terminology and style of the documents you intend to translate.
- e. Category description: Use this field to better describe the particular field or industry in which you're working. For example, if your category is medicine, you might add a particular document, such a surgery, or pediatrics. The description has no influence over the behavior of the Custom Translator or your resulting custom system.
- f. Project label: The [project label](#) distinguishes between projects with the same language pair and category.

As a best practice, use a label *only* if you're planning to build multiple projects for the same language pair and same category and want to access these projects with a different CategoryID. Don't use this field if you're building systems for one category only. A project label is not required and not helpful to distinguish between language pairs. You can use the same label for multiple projects.

Create Project

Project name*

Name your project (max 256 chars)

Description

Add more information to your project (max 500 chars)

Language Pair*

Category*

Category descriptor

Add a category descriptor (max 75 chars)

Project label

Add a label to your project (max 20 chars)

Create

3. Click Create

View project details

The Custom Translator landing page shows the first 10 projects in your workspace. It displays the project name, language pair, category, status, and BLEU score.

After selecting a project, you'll see the following on the project page:

- CategoryID: A CategoryID is created by concatenating the WorkspaceID, project label, and category code. You use the CategoryID with the Text Translator API to get custom translations.
- Train button: Use this button to start a [training a model](#).
- Add documents button: Use this button to [upload documents](#).
- Filter documents button: Use this button to filter and search for specific document(s).

Projects > English to Danish

Category ID: 9e669e56-88b2-4c15-bf1d-b19be66d10d1-GENERAL

Language Pair: English - Danish

Category: General

[Edit project](#)

Data

Models

Create model

Upload files



6 Documents

<input type="checkbox"/>	Name	Document Pairing	Document Type	Language(s)	English Sentences	Danish Sentences
<input type="checkbox"/>	en_dan1	Parallel	Training	English - Danish	60,109	60,109
<input type="checkbox"/>	en_dan2	Parallel	Training	English - Danish	86,980	86,980
<input type="checkbox"/>	en_dan3	Parallel	Training	English - Danish	79,487	79,487
<input type="checkbox"/>	English to Danish Tuning 2	Parallel	Tuning	English - Danish	3,500	3,500
<input type="checkbox"/>	EU_15900000	Parallel	Tuning	Danish - English	4,355	4,355

Next steps

- Learn [how to search, edit, delete project](#).
- Learn [how to upload document](#) to build translation models.

Search, edit, and delete projects

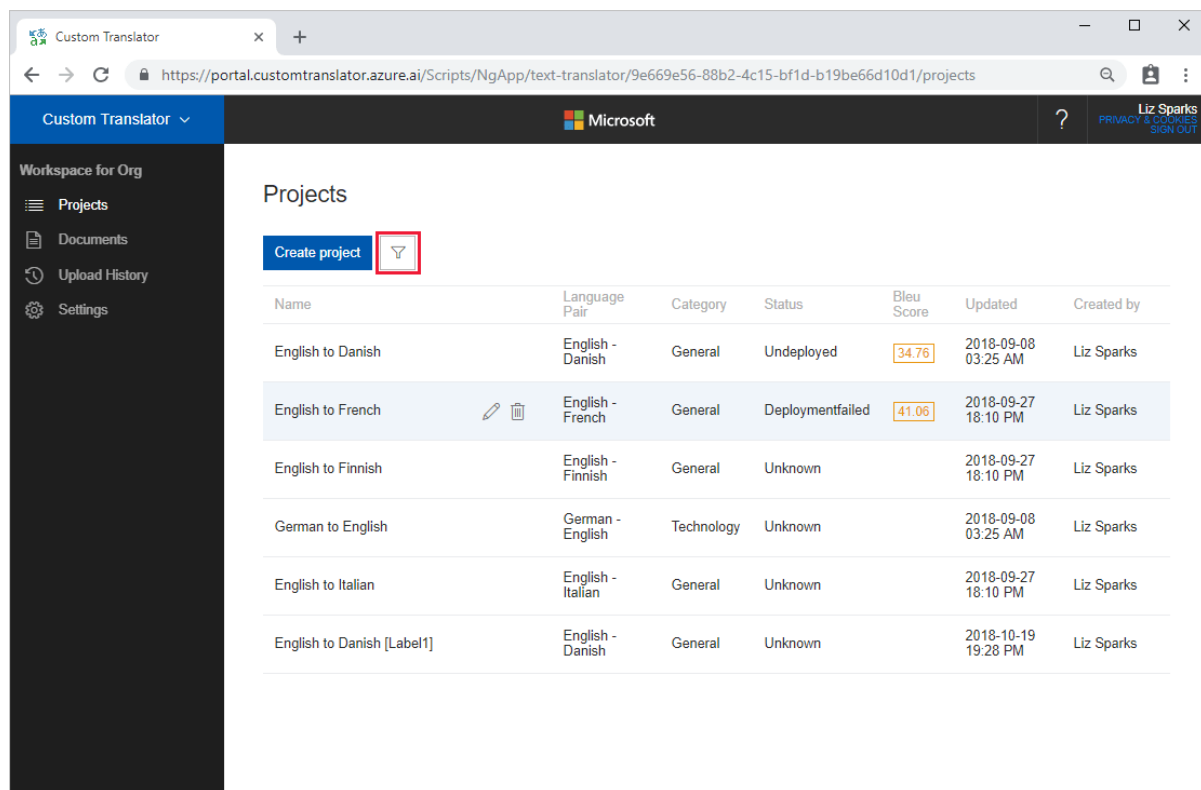
7/28/2019 • 2 minutes to read • [Edit Online](#)

Custom Translator provides multiple ways to manage your projects in efficient manner. You can create many projects, search based on your criteria, and edit your projects. Deleting a project is also possible in Custom Translator.

Search and filter projects

The filter tool allows you to search projects by different filter conditions. It filters like project name, status, source and target language, and category of the project.

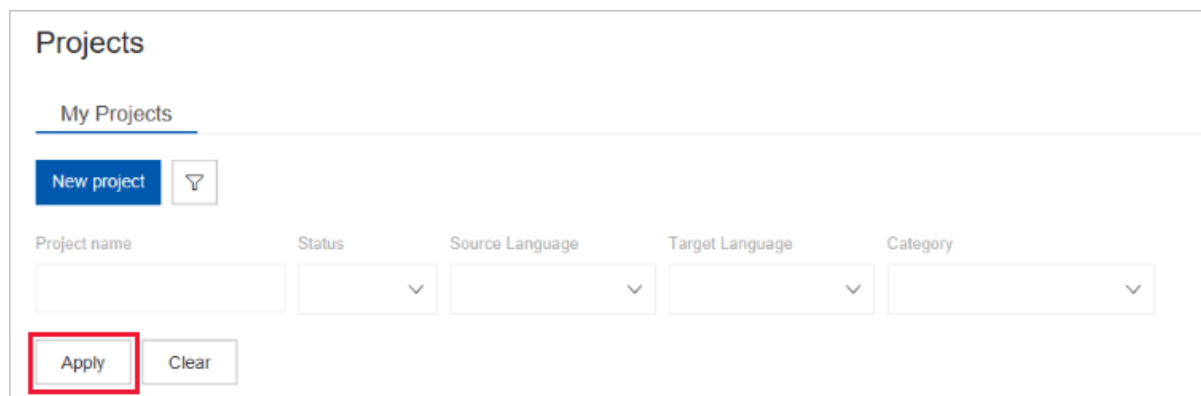
1. Click on the filter button.



The screenshot shows the Custom Translator web interface. On the left is a sidebar with navigation links: Projects, Documents, Upload History, and Settings. The main area is titled 'Projects' and contains a table of project entries. Above the table are two buttons: 'Create project' and a filter icon (a funnel), which is highlighted with a red box. The table has columns for Name, Language Pair, Category, Status, Bleu Score, Updated, and Created by.

Name	Language Pair	Category	Status	Bleu Score	Updated	Created by
English to Danish	English - Danish	General	Undeployed	34.76	2018-09-08 03:25 AM	Liz Sparks
English to French	English - French	General	Deployment failed	41.06	2018-09-27 18:10 PM	Liz Sparks
English to Finnish	English - Finnish	General	Unknown		2018-09-27 18:10 PM	Liz Sparks
German to English	German - English	Technology	Unknown		2018-09-08 03:25 AM	Liz Sparks
English to Italian	English - Italian	General	Unknown		2018-09-27 18:10 PM	Liz Sparks
English to Danish [Label1]	English - Danish	General	Unknown		2018-10-19 19:28 PM	Liz Sparks

2. You can filter by any (or all) of the following fields: project name, status, source language, target language, and category.
3. Click apply.



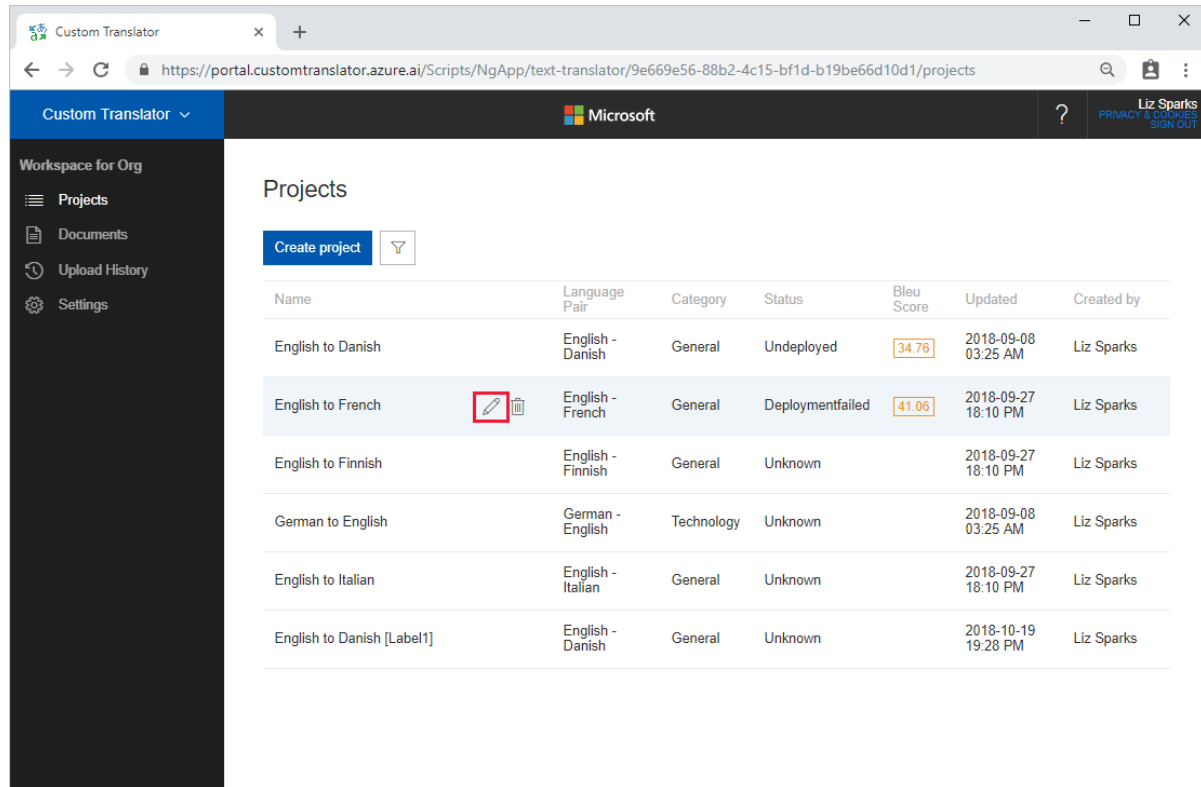
The screenshot shows a filter dialog box titled 'Projects' with a sub-header 'My Projects'. It contains a 'New project' button and a filter icon. Below these are five input fields: 'Project name', 'Status', 'Source Language', 'Target Language', and 'Category'. At the bottom, there are two buttons: 'Apply' (highlighted with a red box) and 'Clear'.

4. Clear the filter to view all your projects by tapping "Clear".

Edit a project

Custom Translator gives you the ability to edit the name and description of a project. Other project metadata like the category, source language, and target language are not available for edit. The steps below describe how to edit a project.

1. Click on the pencil icon that appears when you hover over a project.



2. In the dialog, you can modify either the project name or the description of the project, but cannot modify the project label, category, or language pair.

The 'Edit Project' dialog box has a title bar with a close button (X). It contains two text input fields: 'Project name*' with the value 'English to Finnish' and 'Description' with the value 'This is a test project'. At the bottom right are 'Cancel' and 'Save' buttons.

3. Click on the filter button.

Delete a project

You can delete a project when you no longer need it. Below steps describe how to delete a project.

1. Hover on any project record and click on the trash icon.

Custom Translator

Microsoft

Liz Sparks
PRIVACY & COOKIES
SIGN OUT

Workspace for Org

- Projects
- Documents
- Upload History
- Settings

Projects

Create project

Name	Language Pair	Category	Status	Bleu Score	Updated	Created by
English to Danish	English - Danish	General	Undeployed	34.76	2018-09-08 03:25 AM	Liz Sparks
English to French	English - French	General	Deployment failed	41.06	2018-09-27 18:10 PM	Liz Sparks
English to Finnish	English - Finnish	General	Unknown		2018-09-27 18:10 PM	Liz Sparks
German to English	German - English	Technology	Unknown		2018-09-08 03:25 AM	Liz Sparks
English to Italian	English - Italian	General	Unknown		2018-09-27 18:10 PM	Liz Sparks
English to Danish [Label1]	English - Danish	General	Unknown		2018-10-19 19:28 PM	Liz Sparks

- Confirm deletion. Deleting a project will delete all models that were created within that project. Deleting project will not affect your documents.

Delete project

Are you sure you want to delete this project?

Cancel
Delete

Next steps

- [Upload documents](#) to start building your custom translation model.

Upload a document

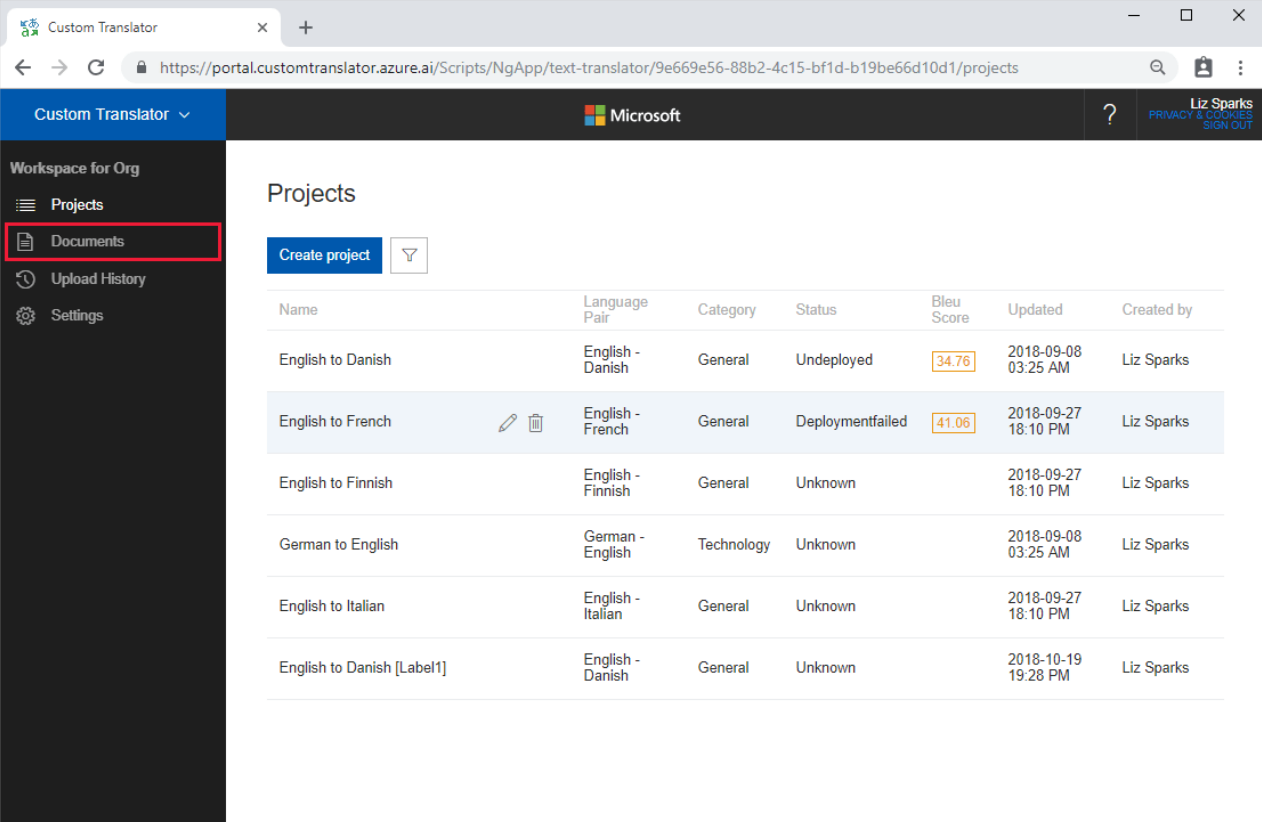
11/5/2019 • 2 minutes to read • [Edit Online](#)

In [Custom Translator](#), you can upload parallel documents to train your translation models. [Parallel documents](#) are pairs of documents where one is a translation of the other. One document in the pair contains sentences in the source language and the other document contains these sentences translated into the target language.

Before uploading your documents, review the [document formats and naming convention guidance](#) to make sure your file format is supported in Custom Translator.

How to upload document?

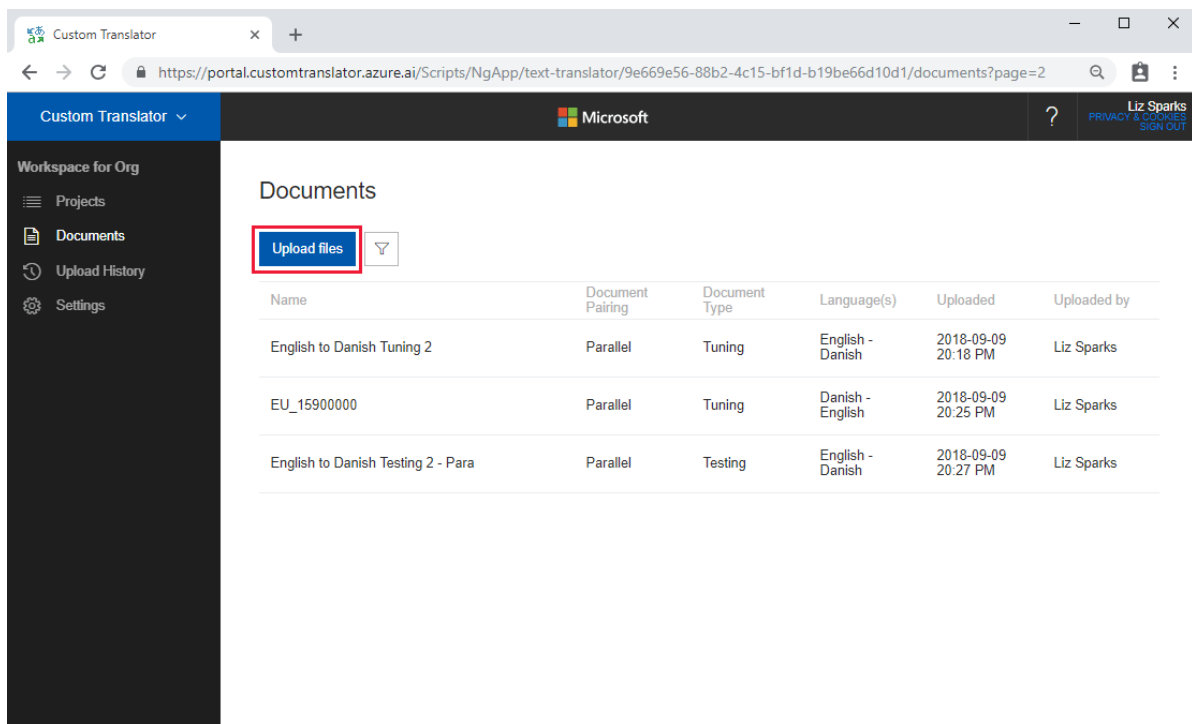
From [Custom Translator](#) portal, click on "Documents" tab to go to documents page.



The screenshot shows the Custom Translator portal interface. The left sidebar has a 'Documents' tab highlighted with a red box. The main content area displays a table of projects. The table has columns for Name, Language Pair, Category, Status, Bleu Score, Updated, and Created by. The 'English to French' project is highlighted in blue and shows a 'Deployment failed' status with a Bleu Score of 41.06. The 'English to Danish' project shows a Bleu Score of 34.76.

Name	Language Pair	Category	Status	Bleu Score	Updated	Created by
English to Danish	English - Danish	General	Undeployed	34.76	2018-09-08 03:25 AM	Liz Sparks
English to French	English - French	General	Deployment failed	41.06	2018-09-27 18:10 PM	Liz Sparks
English to Finnish	English - Finnish	General	Unknown		2018-09-27 18:10 PM	Liz Sparks
German to English	German - English	Technology	Unknown		2018-09-08 03:25 AM	Liz Sparks
English to Italian	English - Italian	General	Unknown		2018-09-27 18:10 PM	Liz Sparks
English to Danish [Label1]	English - Danish	General	Unknown		2018-10-19 19:28 PM	Liz Sparks

1. Click on the Upload files button on the documents page.



2. On the dialog fill in the following information:

a. Document type:

- Training: These document(s) will be used for training set.
- Tuning: These document(s) will be used for tuning set.
- Testing: These document(s) will be used for testing set.
- Phrase Dictionary: These document(s) will be used for phrase dictionary.
- Sentence Dictionary: These document(s) will be used for sentence dictionary.

b. Language pair

c. Override document if exists: Select this check box if you want to overwrite any existing documents with the same name.

d. Fill in the relevant section for either parallel data or combo data.

- Parallel data:
 - Source file: Select source language file from your local computer.
 - Target file: Select target language file from your local computer.
 - Document name: Used only if you're uploading parallel files.
- Combo data:
 - Combo File: Select the combo file from your local computer. Your combo file has both of your source and target language sentences. [Naming convention](#) is important for combo files.

e. Click Upload

Add Documents

*Maximum file size allowed is 100MB

×

Document Type:

Training

Language Pair

English -> German

☐ Override document if it exists

Parallel Data

Source (en) file:

Browse files...

1813n6_en.txt

Target (da) file:

Browse files...

1813n6_de.txt

.TXT|.HTML|.htm|.PDF|.DOCX|.ALIGN file required.

Document Name:

New document

or

Archive or Combo File

Combo file:

Browse files...

.TMX|.XLF|.XLIFF|.LCL|.XLSX|.ZIP file required.

Cancel

Upload

- At this point, we're processing your documents and attempting to extract sentences. You can click "View upload Progress" to check the status of your documents as they process.

Documents processing

×

Your files have been received and are being processed. It may take a while for large files to finish processing.

View upload progress

- This page will display the status, and any errors for each file within your upload. You can view past upload status at any time by clicking on the "Upload history" tab.

Upload History > English to Danish Testing 2 - Para

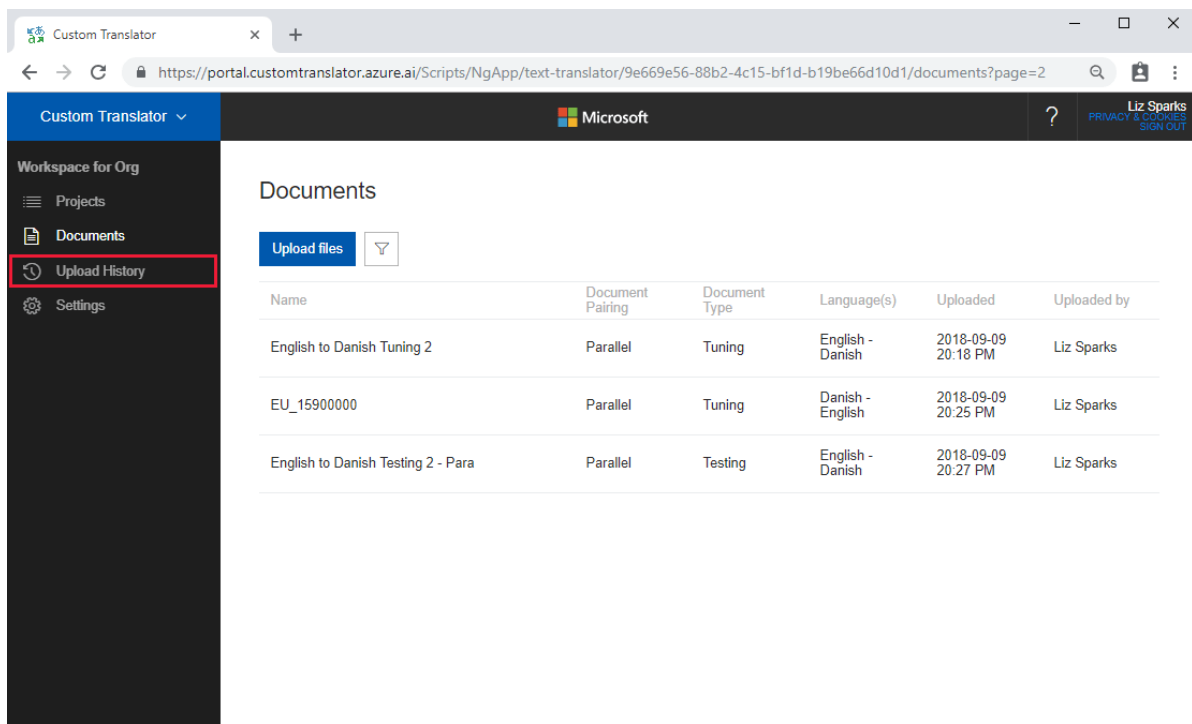
Filter

Name	Status	Language	Error
EU_15900000.enu_en.txt	<div>✓</div> Succeeded	English	
EU_15900000.dan_da.txt	<div>✓</div> Succeeded	Danish	

View upload history

In upload history page you can view history of all document uploads details like document type, language pair, upload status etc.

- From the [Custom Translator](#) portal, click Upload History tab to view history.



- This page shows the status of all of your past uploads. It displays uploads from most recent to least recent. For each upload, it shows the document name, upload status, the upload date, the number of files uploaded, type of file uploaded, and the language pair of the file.

Upload History						
Filter						
Name	Status	Upload Date	Number of Files	Type	Language Pair	Uploaded By
English to Danish Testing 2 - Para	✓ Succeeded	2018-09-09 20:27 PM	2	Testing	Danish, English	Liz Sparks
EU_15900000_tuning.zip	✓ Succeeded	2018-09-09 20:24 PM	2	Tuning	English, Danish	Liz Sparks
EU_15900000_Tuning.zip	✗ 1 error	2018-09-09 20:23 PM	1	Tuning		Liz Sparks
English to Danish Tuning 2	✓ Succeeded	2018-09-09 20:17 PM	2	Tuning	Danish, English	Liz Sparks
doc aug 30 single files end de	✓ Succeeded	2018-08-30 23:46 PM	2	Training	English, German	Liz Sparks
doc aug 30 single files end de	✓ Succeeded	2018-08-30 20:15 PM	2	Training	English, German	Liz Sparks

- Click on any upload history record. In upload history details page, you can view the files uploaded as part of the upload, uploaded status of the file, language of the file and error message (if there is any error in upload).

Next steps

- Use the [document details page](#) to review list of extracted sentences.
- [How to train a model.](#)

View document details

7/28/2019 • 2 minutes to read • [Edit Online](#)

The document list page shows the first 10 document in your workspace. For each of the documents, it displays the name, pairing, type, language, upload time stamp, and the email address of the user who uploaded the document.

Click on an individual document to view the document details page. The document details page displays the list of extracted sentences from the document.

- By default the "source" language is selected in the dropdown field, but you can toggle to view sentences in the target language.
- 20 sentences are displayed per page by default. You can use the pagination control to browse between pages.

Documents > English to Danish Tuning 2

File:

English ▾

20 sentences

En

I am convinced that competitiveness will have greatly increased within 5-10 years and that we will have vehicles with no emissions.

This would be a decisive environmental victory.

In Amendment No 10, something of the utmost importance to all opponents of atomic energy is being put to the vote.

Let the Euratom Treaty be abrogated in its present form but its safety aspects incorporated into the Treaty on European Union.

If that amendment should actually be adopted tomorrow - as I hope - then it would probably be the first time in the history of this Parliament that an anti-atomic amendment reaches the conciliation procedure.

That would mean that, for the first time, the Council would have to come to terms with criticism and rejection of its atomic policy.

There was a similar amendment once before, but that one started from the extremely naive assumption that all Europe's atomic power stations would be shut down by the year 2002.

Anyone who realizes that 34 % of all the electrical power consumed in the EU comes from atomic power stations - in France, the figure is 70 % - will appreciate that this is not an option.

Also, the safety and health provisions must be retained at all costs -if only, of course, because eastward enlargement will bring certain States into the Union whose safety standards are absolutely different from our own.

In that case, of course, they would have to observe those safety provisions.

If all environmentally aware Members, across party boundaries and across national boundaries, too, can find it in their hearts to vote together in tomorrow's vote, then we have a chance to get to the conciliation procedure.


Delete a document

User must be a workspace owner to delete document to delete a document. Additionally, if a document is in use by a model, that is in any part of the training process or any part of the deployment process, the document can't be deleted.

1. Go to document page
2. Hover on any document record and click on the trash icon.

Documents

Upload files

Name		Document Pairing	Document Type	Language(s)	Uploaded
docmay41		Parallel	Training	Russian - English	2018-05-05 01:50 AM
docmay41		Parallel	Training	Russian - English	2018-05-08 20:48 PM
docmay41		Parallel	Training	Russian - English	2018-05-08 21:23 PM
docmay41		Parallel	Training	Russian - English	2018-05-08 21:24 PM
EnglishToItalianParallelTestEnglishToItalian		Parallel	Training	English - Italian	2018-07-17 23:01 PM
en_dan1		Parallel	Training	English - Danish	2018-07-17 23:17 PM
en_dan2		Parallel	Training	English - Danish	2018-07-18 00:33 AM

3. Confirm Delete.

Delete document

×

Are you sure you want to delete this document?

Cancel

Delete

Next steps

- Learn [how to train a model](#).

Train a model

7/28/2019 • 2 minutes to read • [Edit Online](#)

Training a model is the important step to building a translation model, because without a training, model can't be built. Training happens based on documents you select for the trainings.

To train a model:

1. Select the project where you want to build a model.
2. The Data tab for the project will show all the relevant documents for the project language pair. Manually select the documents you want to use to train your model. You can select training, tuning, and testing documents from this screen. Also you just select the training set and have Custom Translator create the tuning and test sets for you.
 - Document name: Name of the document.
 - Pairing: If this document is a parallel or monolingual document. Monolingual documents are currently not supported for training.
 - Document type: Can be training, tuning, testing, or dictionary.
 - Language pair: This show the source and target language for the project.
 - Source sentences: Shows the number of sentences extracted from the source file.
 - Target sentences: Shows the number of sentences extracted from the target file.

Projects > English to Danish

Category ID: 9e669e56-88b2-4c15-bf1d-b19be66d10d1-GENERAL
Language Pair: English - Danish
Category: General
[Edit project](#)

Data

Models

Create model

Upload files

✓ 2 documents selected, 139596 training sentences selected

<input type="checkbox"/>	Name	Document Pairing	Document Type	Language(s)	English Sentences	Danish Sentences
<input checked="" type="checkbox"/>	en_dan1	Parallel	Training	English - Danish	60,109	60,109
<input type="checkbox"/>	en_dan2	Parallel	Training	English - Danish	86,980	86,980
<input checked="" type="checkbox"/>	en_dan3	Parallel	Training	English - Danish	79,487	79,487
<input type="checkbox"/>	English to Danish Tuning 2	Parallel	Tuning	English - Danish	3,500	3,500
<input type="checkbox"/>	EU_15900000	Parallel	Tuning	Danish - English	4,355	4,355
<input type="checkbox"/>	English to Danish Testing 2 - Para	Parallel	Testing	English - Danish	3,050	3,050

3. Click Train button.
4. On the dialog, specify a name for your model.
5. Click Train model.

Train New Model

Model name*

Second model for EN to DAN

Cancel

Train model

6. Custom Translator will submit the training, and show the status of the training in the models tab.

Projects > English to Danish

Category ID: 9e669e56-88b2-4c15-bf1d-b19be66d10d1-GENERAL

Language Pair: English - Danish

Category: General

Edit project

Data

Models

5 models

Name	Status	Bleu Score	Baseline Bleu Score	Training	Dictionary	Tuning	Test	Model Action
English >> Danish 1st model	Undeployed	34.76	34.27	52,015	0	2,500	2,500	Deploy
English >> Danish 2nd model	Succeeded	34.75	34.52	52,015	0	2,500	2,500	Deploy
English >> Danish 3rd model	Succeeded	31	30.88	57,015	0	4,139	2,868	Deploy
Second model for EN to DAN	Succeeded	34.58	34.4	52,015	0	2,500	2,500	Deploy
Model 3 for English to Danish	Submitted			0	0	0	0	

NOTE

Custom Translator supports 10 concurrent trainings within a workspace at any point in time.

Edit a model

You can edit a model using the Edit link on the Model Details page.

1. Click on the Pencil icon.

Projects > English to Danish > English >> Danish 2nd model

Bleu score : 34.75

Baseline Bleu score : 34.52

English - Danish

Edit model

Training Details

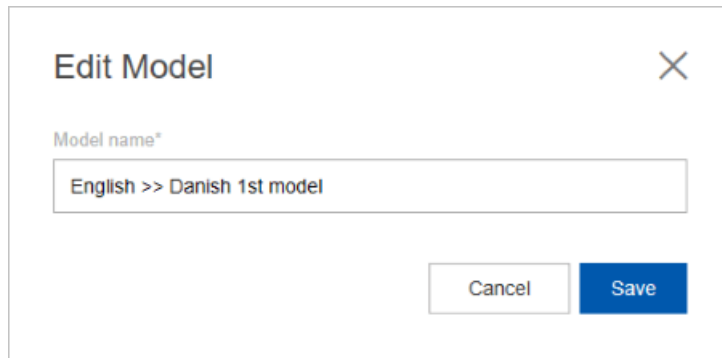
Test

1 documents

Name	Type	English Sentences	Danish Sentences	Aligned Sentences	Used Sentences
<input type="checkbox"/> en_dan1	Training - Parallel	60,109	60,109	57,015	52,015

2. In the dialog change,

- a. Model Name (required): Give your model a meaningful name.



The image shows a modal dialog box titled "Edit Model" with a close button (X) in the top right corner. Inside the dialog, there is a label "Model name*" followed by a text input field containing the text "English >> Danish 1st model". At the bottom right of the dialog, there are two buttons: "Cancel" and "Save".

3. Click Save.

Next steps

- Learn [how to view model details](#).

View model details

7/28/2019 • 3 minutes to read • [Edit Online](#)

The Models tab under project shows all models in that project. All models trained for that project is listed in this tab.

For each model in the project, these details are displayed.

1. **Model Name:** Shows the model name of a given model.
2. **Status:** Shows status of a given model. Your new training will have a status of Submitted until it is accepted. The status will change to Data processing while the service evaluates the content of your documents. When the evaluation of your documents is complete the status will change to Running and you will be able to see the number of sentences that are part of the training, including the tuning and testing sets that are created for you automatically. Below is a list of model status that describes state of the models.
 - **Submitted:** Specifies that the backend is processing the documents for that model.
 - **TrainingQueued:** Specifies that the training is being queued to MT system for that model.
 - **Running:** Specifies that the training is running in MT system for that model.
 - **Succeeded:** Specifies that the training succeeded in MT system and a model is available. In this status, a BLEU score is displayed for that model.
 - **Deployed:** Specifies that the successful trained model is submitted to MT system for deployment.
 - **Undeploying:** Specifies that the deployed model is undeploying.
 - **Undeployed:** Specifies that the undeployment process of a model has been completed successfully.
 - **Training Failed:** Specifies that the training failed. If a training failure occurs, retry the training job. If the error persists, contact us. Don't delete the failed model.
 - **DataProcessingFailed:** Specifies that data processing has failed for one or more documents belonging to the model.
 - **DeploymentFailed:** Specifies that the model deployment has failed.
 - **MigratedDraft:** Specifies that the model is in draft state after migration from Hub to Custom Translator.
3. **BLEU Score:** shows BLEU (Bilingual Evaluation Understudy) score of the model, indicating the quality of your translation system. This score tells you how closely the translations done by the translation system resulting from this training match the reference sentences in the test data set. The BLEU score appears if training is successfully complete. If training isn't complete/ failed, you won't see any BLEU score.
4. **Training Sentence count:** Shows total number of sentences used as training set.
5. **Tuning Sentence count:** Shows total number of sentences used as tuning set.
6. **Testing Sentence count:** Shows total number of sentences used as testing set.
7. **Mono Sentence count:** Shows total number of sentences used as mono set.
8. **Deploy action button:** For a successfully trained model, it shows "Deploy" button if not deployed. If a model is deployed, a "Undeploy" button is shown.

9. Delete: You can use this button if you want to delete the model. Deleting a model won't delete any of the documents used to create that model.

Projects > English to Danish

Category ID: 9e669e56-88b2-4c15-bf1d-b19be66d10d1-GENERAL
Language Pair: English - Danish
Category: General
[Edit project](#)

Data

Models

5 models

Name	Status	Bleu Score	Baseline Bleu Score	Training	Dictionary	Tuning	Test	Model Action
English >> Danish 1st model	Undeployed	34.76	34.27	52,015	0	2,500	2,500	Deploy
English >> Danish 2nd model	Succeeded	34.75	34.52	52,015	0	2,500	2,500	Deploy
English >> Danish 3rd model	Succeeded	31	30.88	57,015	0	4,139	2,868	Deploy
Second model for EN to DAN	Succeeded	34.58	34.4	52,015	0	2,500	2,500	Deploy
Model 3 for English to Danish	Dataprocessing			0	0	0	0	

NOTE

To compare consecutive trainings for the same systems, it is important to keep the tuning set and testing set constant.

View model training details

When your training is complete, you can review details about the training from the details page. Select a project, locate and select the models tab, and choose a model.

The model page has two tabs: Training details and Test.

- Training Details:** This tab shows the list of document(s) used in the training:
 - Documents Name: This field shows the name of the document
 - Document Type: This field shows if this document is parallel/ mono.
 - Sentence count in source language: This field shows number of sentences are there as part of source language.
 - Sentence count in target language: This field shows number of sentences are there as part of target language.
 - Aligned Sentences: This field shows number of sentences has been aligned by Custom Translator during align process.
 - Used Sentences: This field shows number of sentences has been used by Custom Translator during this training.

Projects > English to Danish > English >> Danish 3rd model

Bleu score : 31

Baseline Bleu score : 30.88

English - Danish

[Edit model](#)

Training Details

Test

3 documents

	Name	Type	English Sentences	Danish Sentences	Aligned Sentences	Used Sentences
<input type="checkbox"/>	en_dan1	Training - Parallel	60,109	60,109	57,015	57,015
<input type="checkbox"/>	EU_15900000	Tuning - Parallel	4,355	4,355	4,139	4,139
<input type="checkbox"/>	English to Danish Testing 2 - Para	Testing - Parallel	3,050	3,050	2,868	2,868

2. **Test:** This tab shows the test details for a successful training.

Next steps

- Review [test results](#) and analyze training results.

View system test results

7/28/2019 • 2 minutes to read • [Edit Online](#)

When your training is successful, review system tests to analyze your training results. If you're satisfied with the training results, place a deployment request for the trained model.

System test results page

Select a project, then select the models tab of that project, locate the model you want to use and finally select the test tab.

The test tab shows you:

1. **System Test Results:** The result of the test process in the trainings. The test process produces the BLEU score.

Sentence Count: How many parallel sentences were used in the test set.

BLEU Score: BLEU score generated for a model after training completion.

Status: Indicates if the test process is complete or in progress.

Projects > English to Danish > English >> Danish 1st model

Bleu score : 34.76
Baseline Bleu score : 34.27
English - Danish
[Edit model](#)

Training Details Test

1 tests

Name	Sentence Count	Bleu Score	Status
System Test Results	2500	34.76	Complete

2. Click on the System test results, and that will take you to test result details page. This page shows the machine translation of sentences that were part of the test dataset.
3. The table on the test result details page has two columns - one for each language in the pair. The column for the source language shows the sentence to be translated. The column for the target language contains two sentences in each row.

Ref: This sentence is the reference translation of the source sentence as given in the test dataset.

MT: This sentence is the automatic translation of the source sentence done by the model built after the training was conducted.

Projects > English to Danish > English >> Danish 1st model > System Test Results

English Sentences
Sentence Count: 2500

2,500 test results

English	Danish
The Serbs are not Milosevic, just as the Germans were not Hitler.	Ref: Serberne er ikke Milosevic, lige så lidt som tyskerne var Hitler. MT: Serberne er ikke Milosevic, ligesom tyskerne ikke var Hitler.
When we talk about identification and security services of this type, I hope that we do not limit ourselves and think that it is just a question of numbers.	Ref: Når vi taler om denne type identificerings- og sikkerhedstjenester, håber jeg, at vi ikke begrænser os og synes, at det bare drejer sig om tal. MT: Når vi taler om identifikations- og sikkerhedstjenester af denne type, håber jeg, at vi ikke begrænser os selv og mener, at det kun er et spørgsmål om tal.
It ought only to be permissible for animal feed to contain components of vegetable origin;	Ref: Hvorfor må foder ikke kun bestå af vegetabiliske bestanddele? MT: Det bør kun være tilladt, at foderstoffer indeholder bestanddele af vegetabilisk oprindelse;
Subject: Safe food National campaigns on food safety and health are being conducted in all 15 Member States during the winter and spring.	Ref: Om: Sikker mad I løbet af vinteren og foråret gennemføres der i alle EU's 15 medlemsstater kampagner om fødevarer sikkerhed og sundhed. MT: Om: sikre fødevarer nationale kampagner for fødevarer sikkerhed og sundhed gennemføres i alle 15 medlemsstater i vinter- og forårsperioden.
This proposal is also currently before the Council and Parliament.	Ref: Dette forslag er også på nuværende tidspunkt til behandling i Rådet og Parlamentet. MT: Dette forslag er også i øjeblikket for Rådet og Parlamentet.
I would hope that their names can be added to whatever resolution comes out of today's debate.	Ref: Jeg håber, at deres navne kan blive tilføjet listen over underskrivere af det forslag til beslutning, som måtte blive resultatet af dagens forhandling. MT: Jeg håber, at man kan tilføje deres navne til enhver beslutning, der kommer fra dagens forhandling.

Download test


Click the Download Translations link to download a zip file. The zip contains the machine translations of source sentences in the test data set.

Projects > English to Danish > English >> Danish 1st model

Bleu score : 34.76
Baseline Bleu score : 34.27
English - Danish
[Edit model](#)

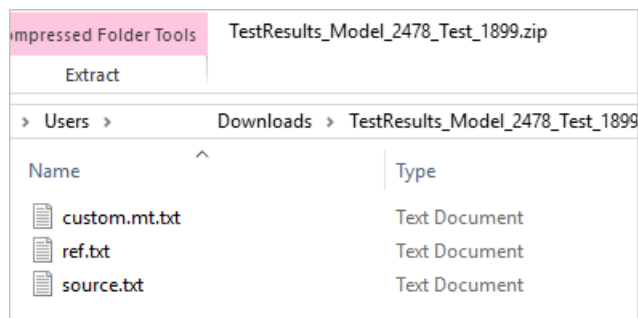
Training Details Test

1 tests

Name		Sentence Count	Bleu Score	Status
System Test Results		2500	34.76	Complete

This downloaded zip archive contains three files.

1. **custom.mt.txt:** This file contains machine translations of source language sentences in the target language done by the model trained with user's data.
2. **ref.txt:** This file contains user provided translations of source language sentences in the target language.
3. **source.txt:** This file contains sentences in the source language.



Deploy a model

To request a deployment:

1. Select a project, go to Models tab.
2. For a successfully trained model, it shows "Deploy" button, if not deployed.

Projects > English to Danish

Category ID: 9e669e56-88b2-4c15-bf1d-b19be6d10d1-GENERAL
Language Pair: English - Danish
Category: General
[Edit project](#)

Data Models

5 models

Name	Status	Bleu Score	Baseline Bleu Score	Training	Dictionary	Tuning	Test	Model Action
English >> Danish 1st model	Undeployed	34.76	34.27	52,015	0	2,500	2,500	Deploy
English >> Danish 2nd model	Succeeded	34.75	34.52	52,015	0	2,500	2,500	Deploy
English >> Danish 3rd model	Succeeded	31	30.88	57,015	0	4,139	2,868	Deploy
Second model for EN to DAN	Succeeded	34.58	34.4	52,015	0	2,500	2,500	Deploy
Model 3 for English to Danish	Dataprocessing			0	0	0	0	

3. Click on Deploy.
4. Select **Deployed** for the region(s) where you want your model to be deployed, and click Save. You can select **Deployed** for multiple regions.

Deploy or undeploy model

Currently all models will be deployed to all regions; however in a future release your model will only be deployed in the regions you have selected.

North America: Undeployed

Europe: Undeployed

Asia Pacific: Undeployed

Cancel Save

5. You can view the status of your model in the "Status" column.

NOTE

Custom Translator supports 10 deployed models within a workspace at any point in time.

Update deployment settings

To update deployment settings:

1. Select a project, and go to the **Models** tab.
2. For a successfully deployed model, it shows an **Update** button.

Projects > English to Danish
[Category ID: 9e669e56-88b2-4c15-bf1d-b19be66d10d1-GENERAL](#)
Language Pair: English - Danish
Category: General
[Edit project](#)

Data		Models						
Name	Status	Bleu Score	Baseline Bleu Score	Training	Dictionary	Tuning	Test	Model Action
English >> Danish 1st model	Deployed	34.76	34.27	52,015	0	2,500	2,500	Update
English >> Danish 2nd model	Succeeded	34.75	34.52	52,015	0	2,500	2,500	
English >> Danish 3rd model	Succeeded	31	30.88	57,015	0	4,139	2,868	
Second model for EN to DAN	Succeeded	34.58	34.4	52,015	0	2,500	2,500	
Model 3 for English to Danish	Trainingfailed			52,421	0	2,500	2,500	

3. Select **Update**.
4. Select **Deployed** or **Undeployed** for the region(s) where you want your model deployed or undeployed, then click **Save**.

Deploy or undeploy model

Currently all models will be deployed to all regions; however in a future release your model will only be deployed in the regions you have selected.

North America:

Deployed

Europe:

Deployed

Asia Pacific:

Deployed

Cancel

Save

NOTE

If you select **Undeployed** for all regions, the model is undeployed from all regions, and put into an undeployed state. It's now unavailable for use.

Next steps

- Start using your deployed custom translation model via [Microsoft Translator Text API V3](#).
- Learn [how to manage settings](#) to share your workspace, manage subscription key.
- Learn [how to migrate your workspace and project](#) from [Microsoft Translator Hub](#)

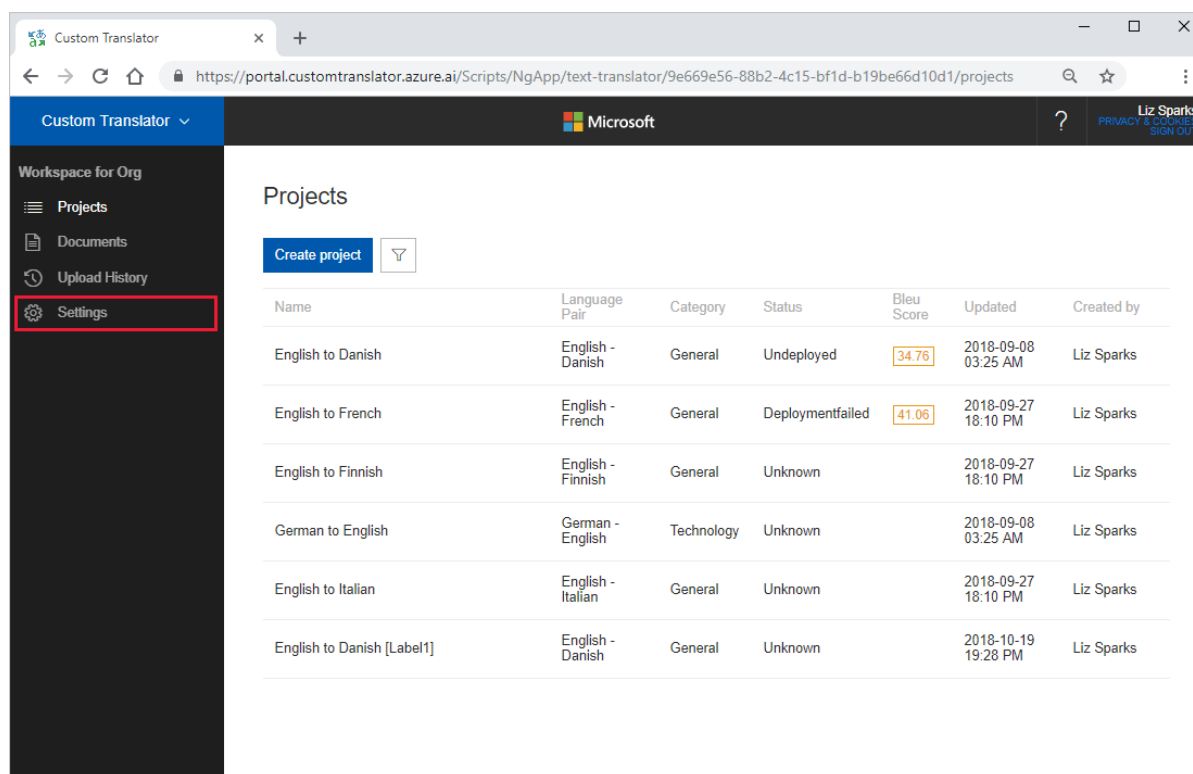
How to manage settings

7/28/2019 • 2 minutes to read • [Edit Online](#)

Within the Custom Translator settings page, you can create a new workspace, share your workspace, and add or modify your Microsoft Translation subscription key.

To access the settings page:

1. Sign in to the [Custom Translator](#) portal.
2. On Custom Translator portal, click on the gear icon in the sidebar.



Associating Microsoft Translator Subscription

You need to have a Microsoft Translator Text API subscription key associated with your workspace to train or deploy models.

If you don't have a subscription, follow the steps below:

1. Subscribe to the Microsoft Translator Text API. This article shows how to subscribe for the Microsoft Translator Text API.
2. Note the key for your translator subscription. Either of the Key1 or Key2 are acceptable.
3. Navigate back to the Custom Translator portal.

Add existing key

1. Navigate to the "Settings" page for your workspace.
2. Click Add Key

Subscription Key

Your Microsoft Translator Text API subscription key is used for billing. Don't have a key? [Create one here.](#)

+ Add key

3. In the dialog, enter the key for your translator subscription, then click the "Add" button.

Add Existing Key

×

Paste your key(s) below

Cancel

Add

4. After you've added a key, you can modify or delete the key at any time.

Subscription Key

Your Microsoft Translator Text API subscription key is used for billing.

HubTextStandardS3: *****16ee4

✎

Change key

🗑

Delete key

Manage your workspace

A workspace is a work area for composing and building your custom translation system. A workspace can contain multiple projects, models, and documents.

If different part of your work needs to be shared with different people, then creating multiple workspaces may be useful.

Create a new workspace

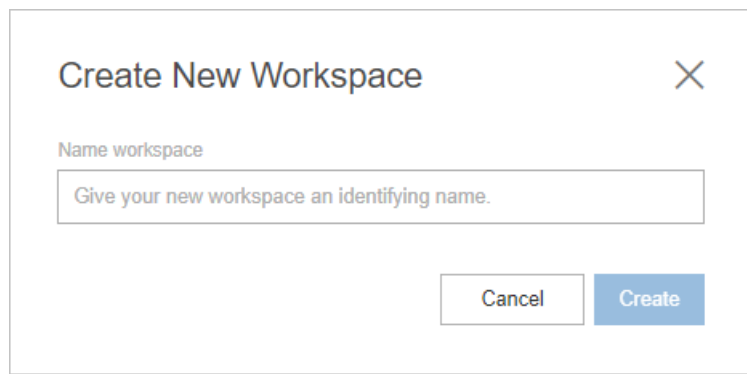
1. Navigate to the workspace "Settings" page.
2. Click on the "New workspace" button in the "Create New Workspace" section.

Create New Workspace

Workspaces are a place for your projects, models, and documents. You can share a workspace with your teammates. If different parts of your work need to be shared with different people, then creating multiple workspaces may be useful.

+ New workspace

3. In the dialog, enter the name of the new workspace.
4. Click "Create".

A dialog box titled "Create New Workspace" with a close button (X) in the top right corner. Below the title is a label "Name workspace" and a text input field with the placeholder text "Give your new workspace an identifying name." At the bottom of the dialog are two buttons: "Cancel" and "Create".

Create New Workspace

Name workspace

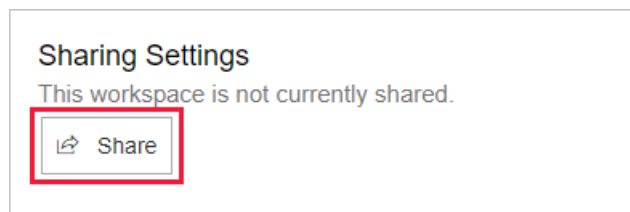
Give your new workspace an identifying name.

Cancel Create

Share your workspace

In Custom Translator you can share your workspace with others, if different part of your work needs to be shared with different people.

1. Navigate to the workspace "Settings" page.
2. Click the "Share" button in the "Sharing Settings" section.

A screenshot of the "Sharing Settings" section. It shows the title "Sharing Settings" and the text "This workspace is not currently shared." Below this text is a button with a share icon and the word "Share", which is highlighted with a red rectangular box.

Sharing Settings

This workspace is not currently shared.

Share

3. On the dialog, enter a comma-separated list of email addresses you want this workspace shared with. Make sure you share with the email address that person uses to sign in to Custom Translator with. Then, select the appropriate level of sharing permission.
4. If your workspace still has the default name "My workspace", you will be required to change it before sharing your workspace.
5. Click "Save".

Sharing permissions

1. **Reader:** A reader in the workspace will be able to view all information in the workspace.
2. **Editor:** An editor in the workspace will be able to add documents, train models, and delete documents and projects. They can add a subscription key, but can't modify who the workspace is shared with, delete the workspace, or change the workspace name.
3. **Owner:** An owner has full permissions to the workspace.


Change sharing permission

When a workspace is shared, the "Sharing Settings" section shows all email addresses that this workspace is shared with. You can change existing sharing permission for each email address if you have owner access to the workspace.

1. In the "Sharing Settings" section for each email a dropdown menu shows the current permission level.
2. Click the dropdown menu and select the new permission level you want to assign to that email address.

Sharing Settings

This workspace is currently shared.

 Add people



Timmy.Forest@contoso.com

Editor ▾



Bret.Clegg@contoso.com

Reader ▾



Next steps

- Learn [how to migrate your workspace and project](#) from [Microsoft Translator Hub](#)

Migrate Hub workspace and projects to Custom Translator

12/10/2019 • 5 minutes to read • [Edit Online](#)

You can easily migrate your [Microsoft Translator Hub](#) workspace and projects to Custom Translator. Migration is initiated from Microsoft Hub by selecting a workspace or project, then selecting a workspace in Custom Translator, and then selecting the trainings you want to transfer. After the migration starts, the selected training settings will be transferred with all relevant documents. Deployed models are trained and can be autodeployed upon completion.

These actions are performed during migration:

- All documents and project definitions will have their names transferred with the addition of "hub_" prefixed to the name. Auto-generated test and tuning data will be named `hub_systemtune_<modelid>` or `hub_systemtest_<modelid>`.
- Any trainings that were in the deployed state when the migration takes place will automatically be trained using the documents of the Hub training. This training will not be charged to your subscription. If auto-deploy was selected for the migration, the trained model will be deployed upon completion. Regular hosting charges will be applied.
- Any migrated trainings that were not in the deployed state will be put into the migrated draft state. In this state, you will have the option of training a model with the migrated definition, but regular training charges will apply.
- At any point, the BLEU score migrated from the Hub training can be found in the TrainingDetails page of the model in the "Bleu score in MT Hub" heading.

NOTE

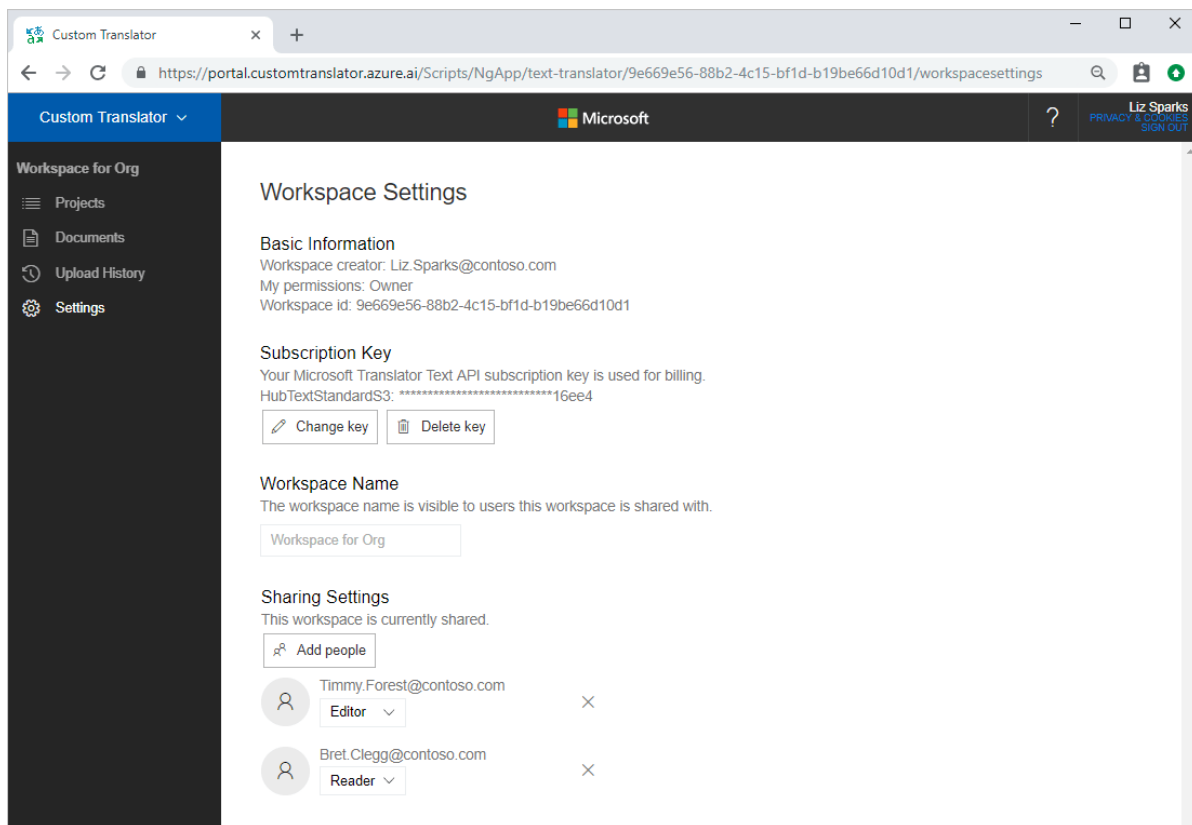
For a training to succeed, Custom Translator requires a minimum of 10,000 unique extracted sentences. Custom Translator can't conduct a training with fewer than the [suggested minimum](#).

Find Custom Translator Workspace ID

To migrate [Microsoft Translator Hub](#) workspace, you need destination Workspace ID in Custom Translator. The destination workspace in Custom Translator is where all your Hub workspaces and projects shall be migrated to.

You will find your destination Workspace ID on Custom Translator Settings page:

1. Go to "Setting" page in the Custom Translator portal.
2. You will find the Workspace ID in the Basic Information section.



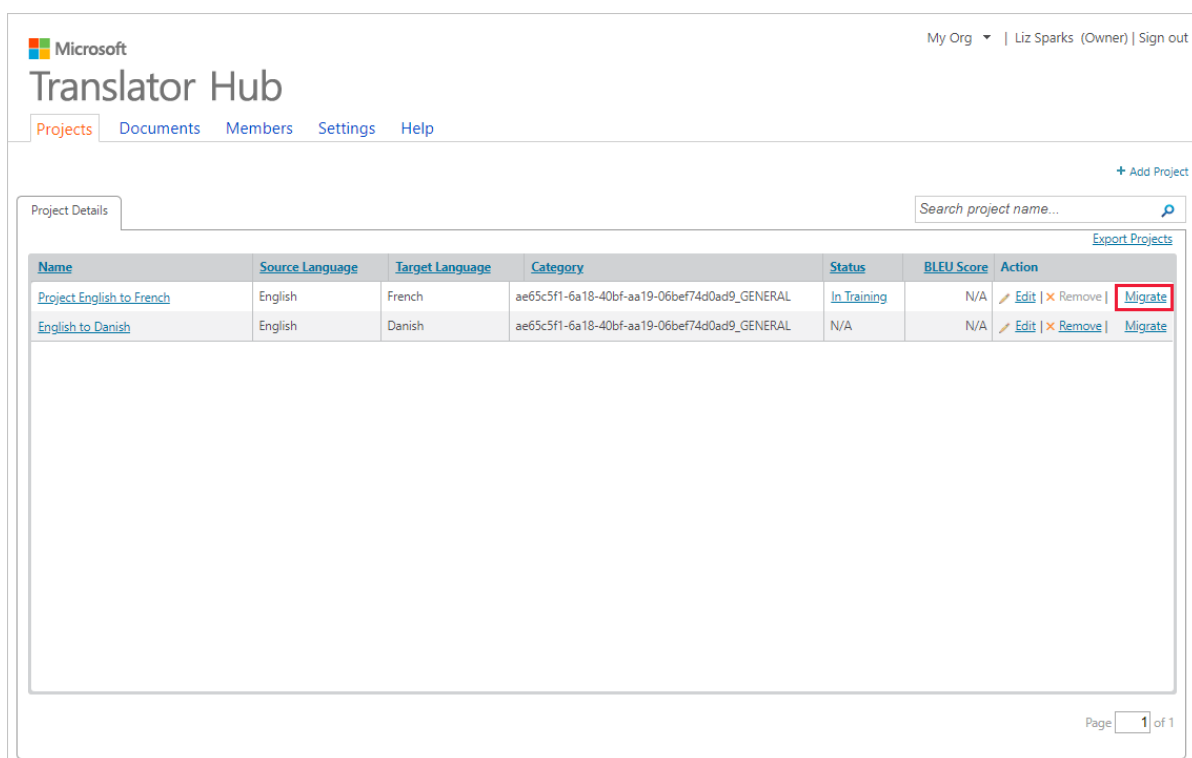
3. Keep your destination Workspace ID to refer during the migration process.

Migrate a project

If you want to migrate your projects selectively, Microsoft Translator Hub gives you that ability.

To migrate a project:

1. Sign in to Microsoft Translator Hub.
2. Go to "Projects" page.
3. Click "Migrate" link for appropriate project.




4. Upon pressing the migrate link you will be presented with a form allowing you to:
 - Specify the workspace you wish to transfer to on Custom Translator
 - Indicate whether you wish to transfer all trainings with successful trainings or just the deployed trainings. By default all successful trainings will be transferred.
 - Indicate whether you would like your training auto deployed when training completes. By default your training will not be auto deployed upon completion.
5. Click "Submit Request".

Migrate a workspace

In addition to migrating a single project, you may also migrate all projects with successful trainings in a workspace. This will cause each project in the workspace to be evaluated as though the migrate link had been pressed. This feature is suitable for users with many projects who want to migrate all of them to Custom Translator with the same settings. A workspace migration can be initiated from the settings page of Translator Hub.

To migrate a workspace:

1. Sign in to Microsoft Translator Hub.
2. Go to "Settings" page.
3. On "Settings" page click "Migrate Workspace data to Custom Translator".



Translator Hub

[Projects](#) [Documents](#) [Members](#) [Settings](#) [Help](#)

Workspace Name:*

Dictionary only testing

Workspace ID:

a32f0eea-cd7b-44ca-9b36-50cb4fb10bf3

Organization:

Email:*

Liz.Sparks@contoso.com?

Keep Data Private:

☐?

Subscription Key:

?

*Required

Save Changes

Associate or Start a Microsoft Translator API subscription

Associating a Microsoft Translator subscription unlocks these features:

- Access to community and collaborative translations
- Import community and collaborative translations as training data
- Preauthorize higher levels of translation volumes

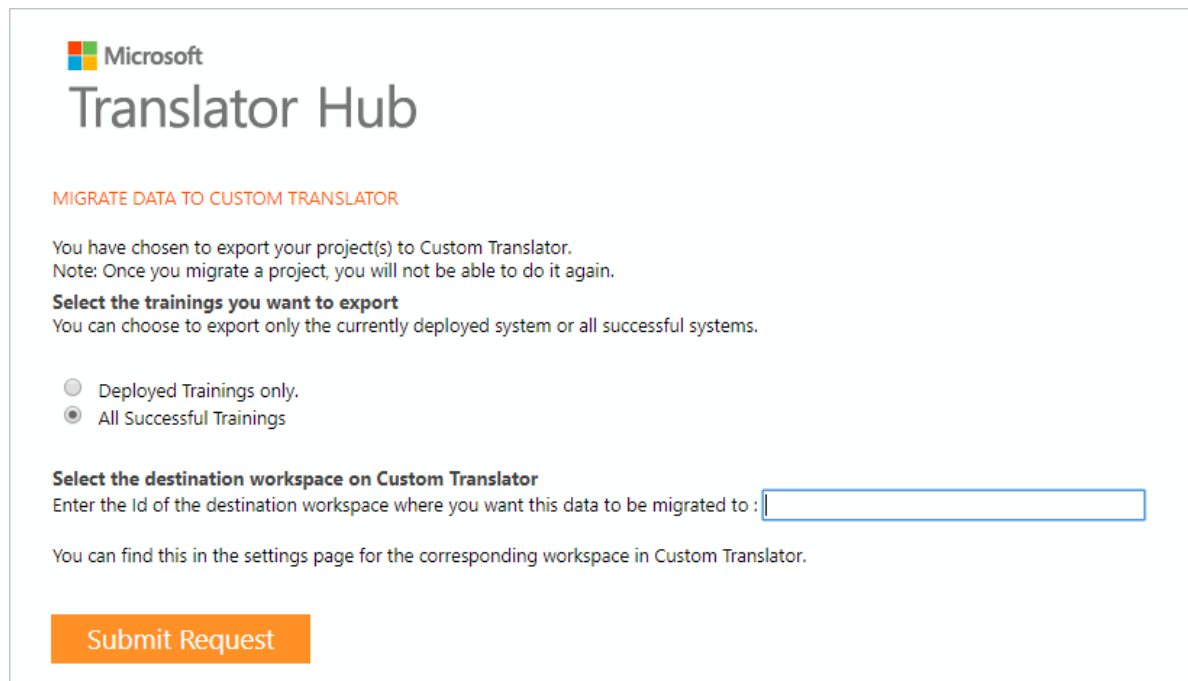
[Create new workspace / Join an existing workspace](#)

[Delete this workspace](#)

Migrate Workspace data to Custom Translator

4. On the next page select either of these two options:

- a. Deployed Trainings only: Selecting this option will migrate only your deployed systems and related documents.
- b. All Successful Trainings: Selecting this option will migrate all your successful trainings and related documents.
- c. Enter your destination Workspace ID in Custom Translator.



The screenshot shows the Microsoft Translator Hub interface for migrating data to Custom Translator. It includes the Microsoft logo and the title 'Translator Hub'. The main heading is 'MIGRATE DATA TO CUSTOM TRANSLATOR'. Below this, a note states: 'You have chosen to export your project(s) to Custom Translator. Note: Once you migrate a project, you will not be able to do it again.' The section 'Select the trainings you want to export' offers two radio button options: 'Deployed Trainings only.' and 'All Successful Trainings', with the latter being selected. A sub-section 'Select the destination workspace on Custom Translator' contains a text input field for the workspace ID, with a note below it: 'Enter the Id of the destination workspace where you want this data to be migrated to :'. A final note states: 'You can find this in the settings page for the corresponding workspace in Custom Translator.' At the bottom is an orange 'Submit Request' button.

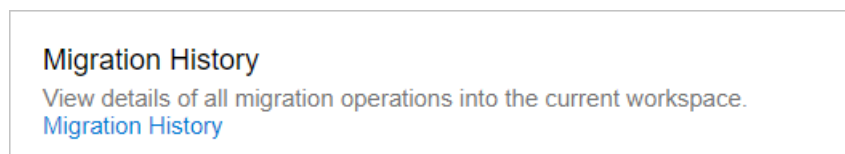
5. Click Submit Request.

Migration History

When you have requested workspace/ project migration from Hub, you'll find your migration history in Custom Translator Settings page.

To view the migration history, follow these steps:

- 1. Go to "Setting" page in the Custom Translator portal.
- 2. In the Migration History section of the Settings page, click Migration History.



Migration History page displays following information as summary for every migration you requested.

- 1. Migrated By: Name and email of the user submitted this migration request
- 2. Migrated On: Date and time stamp of the migration
- 3. Projects: Number of projects requested for migration v/s number of projects successfully migrated.
- 4. Trainings: Number of trainings requested for migration v/s number of trainings successfully migrated.
- 5. Documents: The number of documents requested for migration v/s number of documents successfully migrated.

Workspace Settings > Migration History

Migrated By	Migrated On	Projects	Trainings	Documents	Details
Liz Sparks (Liz.Sparks@contoso.com)	2018-10-30 01:34 AM	Succeeded: 1 / 1	Succeeded: 9 / 9	Succeeded: 17 / 17	Export Details

If you want more detailed migration report about your projects, trainings and documents, you have option export details as CSV.

Implementation Notes

- Systems with language pairs NOT yet available in Custom Translator will only be available to access data or undeploy through Custom Translator. These projects will be marked as "Unavailable" on the Projects page. As we enable new language pairs with Custom Translator, the projects will become active to train and deploy.
- Migrating a project from Hub to Custom Translator will not have any impact on your Hub trainings or projects. We do not delete projects or documents from Hub during a migration and we do not undeploy models.
- You are only permitted to migrate once per project. If you need to repeat a migration on a project, please contact us.
- Custom Translator supports NMT language pairs to and from English. [View the complete list of supported languages](#). Hub does not require baseline models and therefore supports several thousand languages. You can migrate an unsupported language pair, however we will only perform the migration of documents and project definitions. We will not be able to train the new model. Furthermore, these documents and projects will be displayed as inactive in order to indicate that they can't be used at this time. If support is added for these projects and/or documents, they will become active and trainable.
- Custom Translator does not currently support monolingual training data. Like unsupported language pairs, you can migrate monolingual documents, but they show as inactive until monolingual data is supported.
- Custom Translator requires 10k parallel sentences in order to train. Microsoft Hub could train on a smaller set of data. If a training is migrated which does not meet this requirement, it will not be trained.

Custom Translator versus Hub

This table compares the features between Microsoft Translator Hub and Custom Translator.

	HUB	CUSTOM TRANSLATOR
Customization feature status	General Availability	General Availability
Text API version	V2	V3
SMT customization	Yes	No
NMT customization	No	Yes
New unified Speech services customization	No	Yes
No Trace	Yes	Yes

New languages

If you are a community or organization working on creating a new language system for Microsoft Translator,

reach out to custommt@microsoft.com for more information.

Next steps

- [Train a model](#).
- Start using your deployed custom translation model via [Microsoft Translator Text API V3](#).

Unsupported language deployments

11/8/2019 • 2 minutes to read • [Edit Online](#)

With the upcoming retirement of the Microsoft Translator Hub, Microsoft will be undeploying all models currently deployed through the Hub. Many of you have models deployed in the Hub whose language pairs are not supported in Custom Translator. We do not want users in this situation to have no recourse for translating their content.

We now have a process that allows you to deploy your unsupported models through the Custom Translator. This process enables you to continue to translate content using the latest V3 API. These models will be hosted until you choose to undeploy them or the language pair becomes available in Custom Translator. This article explains the process to deploy models with unsupported language pairs.

Prerequisites

In order for your models to be candidates for deployment, they must meet the following criteria:

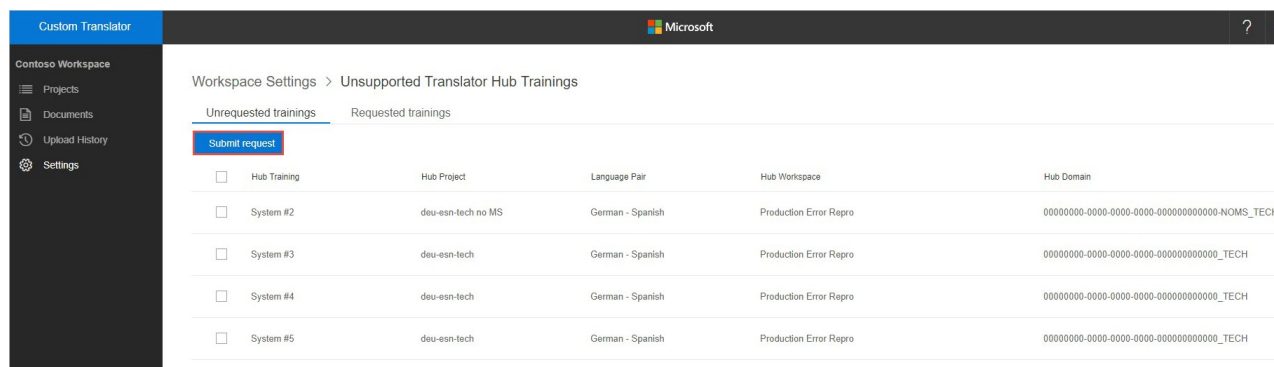
- The project containing the model must have been migrated from the Hub to the Custom Translator using the Migration Tool. The process for migrating projects and workspaces can be found [here](#).
- The model must be in the deployed state when the migration happens.
- The language pair of the model must be an unsupported language pair in Custom Translator. Language pairs in which a language is supported to or from English, but the pair itself does not include English, are candidates for unsupported language deployments. For example, a Hub model for a French to German language pair is considered an unsupported language pair even though French to English and English to German are supported language pair.

Process

Once you have migrated models from the Hub that are candidates for deployment, you can find them by going to the **Settings** page for your workspace and scrolling to the end of the page where you will see an **Unsupported Translator Hub Trainings** section. This section only appears if you have projects that meet the prerequisites mentioned above.

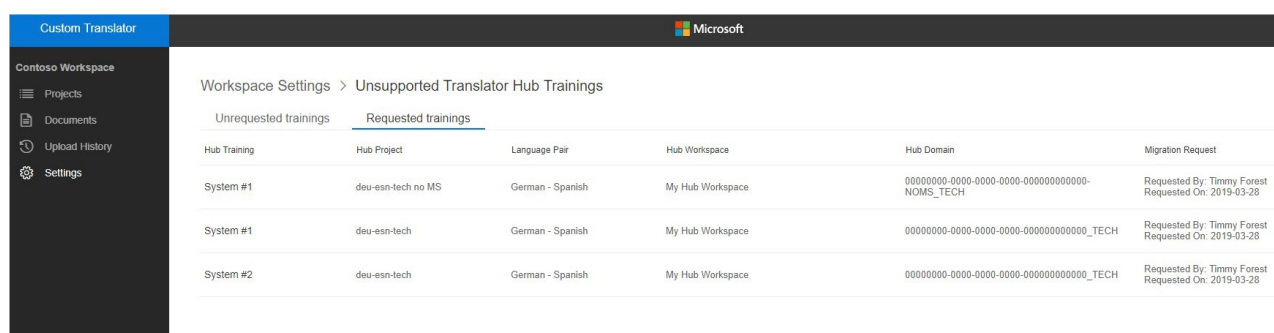
The screenshot shows the 'Custom Translator' interface with the 'Contoso Workspace' selected. The left sidebar contains a 'Settings' link highlighted with a red box. The main content area is titled 'Workspace Settings' and includes sections for 'Basic Information', 'Sharing Settings', 'Create New Workspace', and 'Migration History'. At the bottom, a section titled 'Unsupported Translator Hub Trainings' is highlighted with a red box. This section contains a paragraph explaining that Hub trainings using unsupported language pairs can be marked for use with the Translator Text API V3 by April 30, 2019, and that these trainings will be available for use with the API by June 15, 2019. It also notes that a local dialect of a supported language is considered a supported language.

Within the **Unsupported Translator Hub Trainings** selection page, the **Unrequested trainings** tab contains models that are eligible for deployment. Select the models you wish to deploy and submit a request. Before the April 30 deployment deadline, you can select as many models as you wish for deployment.



<input type="checkbox"/>	Hub Training	Hub Project	Language Pair	Hub Workspace	Hub Domain
<input type="checkbox"/>	System #2	deu-esn-tech no MS	German - Spanish	Production Error Repro	00000000-0000-0000-0000-000000000000-NOMS_TECH
<input type="checkbox"/>	System #3	deu-esn-tech	German - Spanish	Production Error Repro	00000000-0000-0000-0000-000000000000_TECH
<input type="checkbox"/>	System #4	deu-esn-tech	German - Spanish	Production Error Repro	00000000-0000-0000-0000-000000000000_TECH
<input type="checkbox"/>	System #5	deu-esn-tech	German - Spanish	Production Error Repro	00000000-0000-0000-0000-000000000000_TECH

Once submitted, the model will no longer be available on the **Unrequested trainings** tab and will instead appear on the **Requested trainings** tab. You can view your requested trainings at any time.



Hub Training	Hub Project	Language Pair	Hub Workspace	Hub Domain	Migration Request
System #1	deu-esn-tech no MS	German - Spanish	My Hub Workspace	00000000-0000-0000-0000-000000000000-NOMS_TECH	Requested By: Timmy Forest Requested On: 2019-03-28
System #1	deu-esn-tech	German - Spanish	My Hub Workspace	00000000-0000-0000-0000-000000000000_TECH	Requested By: Timmy Forest Requested On: 2019-03-28
System #2	deu-esn-tech	German - Spanish	My Hub Workspace	00000000-0000-0000-0000-000000000000_TECH	Requested By: Timmy Forest Requested On: 2019-03-28

What's next?

The models you selected for deployment are saved once the Hub is decommissioned and all models are undeployed. You have until May 24 to submit requests for deployment of unsupported models. We will deploy these models on June 15 at which point they will be accessible through the Translator V3 API. In addition, they will be available through the V2 API until July 1.

For further information on important dates in the deprecation of the Hub check [here](#). Once deployed, normal hosting charges will apply. See [pricing](#) for details.

Unlike standard Custom Translator models, Hub models will only be available in a single region, so multi-region hosting charges will not apply. Once deployed, you will be able to undeploy your Hub model at any time through the migrated Custom Translator project.

Next steps

- [Train a model](#).
- Start using your deployed custom translation model via [Microsoft Translator Text API V3](#).

Custom Translator Glossary

11/8/2019 • 2 minutes to read • [Edit Online](#)

The [Custom Translator](#) glossary explains terms that you might encounter as you work with the Custom Translator.

WORD OR PHRASE	DEFINITION
Source Language	The source language is the language you are starting with and want to convert to another language (the "target").
Target Language	The target language is the language that you want the machine translation to provide after it receives the source language.
Monolingual File	A monolingual file has a single language that is not paired with another file of a different language.
Parallel Files	A parallel file is combination of two files with corresponding text. One file has the source language. The other has the target language.
Sentence Alignment	Parallel dataset must have aligned sentences to sentences that represent the same text in both languages. For instance, in a source parallel file the first sentence should, in theory, map to the first sentence in the target parallel file.
Aligned Text	One of the most important steps of file validation is to align the sentences in the parallel documents. Things are expressed differently in different languages. Also different languages have different word orders. This step does the job of aligning the sentences with the same content so that they can be used for training. A low sentence alignment indicates there might be something wrong with one or both of the files.
Word Breaking/ Unbreaking	Word breaking is the function of marking the boundaries between words. Many writing systems use a space to denote the boundary between words. Word unbreaking refers to the removal of any visible marker that may have been inserted between words in a preceding step.
Delimiters	Delimiters are the ways that a sentence is divided up into segments or delimit the margin between sentences. For instance, in English spaces delimit words, colons, and semi-colons delimit clauses and periods delimit sentences.
Training Files	A training file is used to teach the machine translation system how to map from one language (the source) to a target language (the target). The more data you can provide the better the system will perform at translation.

WORD OR PHRASE	DEFINITION
Tuning Files	These files are often randomly derived from the training set (if you do not select any tuning set). The sentences autoselected are used to tune up the system and make sure that it is functioning properly. Should you decide to create your own Tuning files, make sure they are a random set of sentences across domains if you wish to create a general-purpose translation model.
Testing Files	These files are often derived files, randomly selected from the training set (if you do not select any test set). The purpose of these sentences is to evaluate the translation model's accuracy. These are sentences you want to make sure the system accurately translates. So you may wish to create a testing set and upload it to the translator to ensure that these sentences are used in the system's evaluation (the generation of a BLEU score).
Combo file	A type of file in which the source and translated sentences are contained in the same file. Supported file formats (".tmx", ".xliff", ".xlf", ".lcl", ".xlsx").
Archive file	A file that contains other files. Supported file formats (zip, gz, tgz).
BLEU Score	BLEU is the industry standard method for evaluating the "precision" or accuracy of the translation model. Though other methods of evaluation exist, Microsoft Translator relies BLEU method to report accuracy to Project Owners.

Custom Translator frequently asked questions

11/8/2019 • 2 minutes to read • [Edit Online](#)

This article contains answers to frequently asked questions about [Custom Translator](#).

What are the current restrictions in Custom Translator?

There are restrictions and limits with respect to file size, model training, and model deployment. Keep these restrictions in mind when setting up your training to build a model in Custom Translator.

- Submitted files must be less than 100 MB in size.
- Monolingual data is not supported.

When should I request deployment for a translation system that has been trained?

It may take several trainings to create the optimal translation system for your project. You may want to try using more training data or more carefully filtered data, if the BLEU score and/ or the test results are not satisfactory. You should be strict and careful in designing your tuning set and your test set, to be fully representative of the terminology and style of material you want to translate. You can be more liberal in composing your training data, and experiment with different options. Request a system deployment when you are satisfied with the translations in your system test results, have no more data to add to the training to improve your trained system, and you want to access the trained model via APIs.

How many trained systems can be deployed in a project?

Only one trained system can be deployed per project. It may take several trainings to create a suitable translation system for your project and we encourage you to request deployment of a training that gives you the best result. You can determine the quality of the training by the BLEU score (higher is better), and by consulting with reviewers before deciding that the quality of translations is suitable for deployment.

When can I expect my trainings to be deployed?

The deployment generally takes less than an hour.

How do you access a deployed system?

Deployed systems can be accessed via the Microsoft Translator Text API V3 by specifying the CategoryID. More information about the Translator Text API can be found in the [API Reference](#) webpage.

How do I skip alignment and sentence breaking if my data is already sentence aligned?

The Custom Translator skips sentence alignment and sentence breaking for TMX files and for text files with the `.align` extension. `.align` files give users an option to skip Custom Translator's sentence breaking and alignment process for the files that are perfectly aligned, and need no further processing. We recommend using `.align` extension only for files that are perfectly aligned.

If the number of extracted sentences does not match the two files with the same base name, Custom Translator will still run the sentence aligner on `.align` files.

I tried uploading my TMX, but it says "document processing failed".

Ensure that the TMX conforms to the TMX 1.4b Specification at <https://www.gala-global.org/tmx-14b>.