Amazon Translate

Developer Guide

Amazon Translate: Developer Guide

The AWS Documentation website is getting a new look!

Try it now and let us know what you think. Switch to the new look >>

You can return to the original look by selecting English in the language selector above.

Table of Contents

What Is Amazon Translate?

Amazon Translate is a text translation service that uses advanced machine learning technologies to provide high-quality translation on demand. You can use Amazon Translate to translate unstructured text documents or to build applications that work in multiple languages.

Supported Languages and Language Codes

Amazon Translate provides translation between a source language (the input language) and a target language (the output language). A source language-target language combination is known as a *language pair*.

Amazon Translate can translate text between the languages listed in the following table, with a few exceptions noted below.

Language	Language Code
Arabic	ar
Chinese (Simplified)	zh
Chinese (Traditional)	zh-TW
Czech	cs
Danish	da
Dutch	nl
English	en
Finnish	fi
French	fr
German	de
Greek	el
Hebrew	he
Hindi	hi
Hungarian	hu
Indonesian	id
Italian	it
Japanese	ja
Korean	ko
Malay	ms

Language	Language Code
Norwegian	no
Persian	fa
Polish	pl
Portuguese	pt
Romanian	ro
Russian	ru
Spanish	es
Swedish	sv
Thai	th
Turkish	tr
Ukrainian	uk
Urdu	ur
Vietnamese	vi

Amazon Translate can't translate text between the following language pairs:

- Chinese (Simplified) to Chinese (Traditional)
- Chinese (Traditional) to Chinese (Simplified)
- Korean to Hebrew
- Norwegian to Arabic
- Norwegian to Hebrew

Use Cases

Use Amazon Translate to do the following::

Enable multilingual user experiences in your applications by integrating Amazon Translate:

- Translate company-authored content, such as meeting minutes, technician reports, knowledge-base articles, posts, and more.
- Translate interpersonal communications, such as email, in-game chat, customer service chat, and more to enable customers and employees to connect in their preferred language.

Process and manage your company's incoming data:

- Analyze text, such as social media and news feeds, in many languages.
- Search for information, such as for eDiscovery cases, in many languages.

Enable language-independent processing by integrating Amazon Translate with other AWS services:.

- Extract named entities, sentiment, and key phrases from unstructured text, such as social media streams with Amazon Comprehend.
- Make subtitles and live captioning available in many languages with Amazon Transcribe.
- Speak translated content with Amazon Polly.

- Translate document repositories stored in Amazon S3 .
- Translate text stored in the following databases: Amazon DynamoDB, Amazon Aurora, and Amazon Redshift.
- Seamlessly integrate workflows with AWS Lambda or AWS Glue.

Are You a First-time User of Amazon Translate?

If you are a first-time user, we recommend that you read the following sections in order:

1. How Amazon Translate Works (p. 4) – Introduces Amazon Translate.
2. Getting Started with Amazon Translate (p. 6) – Explains how to set up your AWS account and start using Amazon Translate.
3. Examples (p. 22) – Provides code examples in Java and Python. Use the examples to explore how Amazon Translate works.
4. API Reference (p. 71) – Contains reference documentation for Amazon Translate operations.

Amazon Translate Pricing

As with other AWS products, there are no contracts or minimum commitments for using Amazon Translate. For more information about the cost of using Amazon Translate, see Amazon Translate Pricing.

How Amazon Translate Works

The Amazon Translate service is based on neural networks trained for language translation. This enables you to translate between a source language (the original language of the text being translated) and a target language (the language into which the text is being translated). It is not necessary to use English as either the source or the target language but not all language combinations are supported by Amazon Translate. For more information, see Supported Languages and Language Codes (p. 1).

When working with Amazon Translate, you will provide source text and get output text:

- **Source text**—The text that you want to translate. You provide the source text in UTF-8 format.
- **Output text**—The text that Amazon Translate has translated into the target language. Output text is also in UTF-8 format. Depending on the source and target languages, there might be more characters in the output text than in the input text.

The translation model has two components, the encoder and the decoder. The *encoder* reads a source sentence one word at a time and constructs a semantic representation that captures its meaning. The *decoder* uses the semantic representation to generate a translation one word at a time in the target language.

Amazon Translate uses attention mechanisms to understand context. This helps it decide which words in the source text are most relevant for generating the next target word. Attention mechanisms enable the decoder to focus on the most relevant parts of a source sentence. This ensures that the decoder correctly translates ambiguous words or phrases.

The target word that the model generates becomes input to the decoder. The network continues generating words until it reaches the end of the sentence.

Automatic Language Detection

Amazon Translate can automatically detect the language used in your source text. To use automatic language detection, specify `auto` as the source language. Amazon Translate calls Amazon Comprehend on your behalf to determine the language used in the source text. By choosing automatic language detection, you agree to the service terms and agreements for Amazon Comprehend. For information about pricing for Amazon Comprehend, see Amazon Comprehend Pricing.

Exception Handling

If you specify a source or target language that isn't supported, Amazon Translate returns the following exceptions:

- **UnsupportedLanguagePairException** – Amazon Translate supports translation between the supported languages. This exception is returned if translation between the language pair is unsupported. For more information, see Supported Languages (p. 1).
- **DetectedLanguageLowConfidenceException** – If you use automatic language detection, and Amazon Translate has low confidence that it detected the correct source language, it returns this exception. If a low confidence level is acceptable, you can use the source language returned in the exception.

Next Steps

Now that you've learned how Amazon Translate works, you can explore the following sections to learn about creating a solution.

- Getting Started with Amazon Translate (p. 6)
- Examples (p. 22)

Getting Started with Amazon Translate

To get started using Amazon Translate, set up an AWS account and create an AWS Identity and Access Management (IAM) user. To use the AWS Command Line Interface (AWS CLI), download and configure it.

Topics

Step 1: Set Up an AWS Account and Create an Administrator User

Before you use Amazon Translate for the first time, complete the following tasks:

1. Sign Up for AWS (p. 6)
2. Create an IAM User (p. 6)

Sign Up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including Amazon Translate. You are charged only for the services that you use.

With Amazon Translate, you pay only for the resources that you use. If you are a new AWS customer, you can get started with Amazon Translate for free. For more information, see AWS Free Usage Tier.

If you already have an AWS account, skip to the next section.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Record your AWS account ID because you'll need it for the next task.

Create an IAM User

AWS services, such as Amazon Translate, require that you provide credentials when you access them. This allows the service to determine whether you have permissions to access the service's resources.

We strongly recommend that you access AWS using AWS Identity and Access Management (IAM), not the credentials for your AWS account. To use IAM to access AWS, create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user. You can then access AWS using a special URL and the IAM user's credentials.

Exercises in this guide assume that you have an IAM user with administrator privileges called `adminuser`.

To create an administrator user

- In your AWS account, create an administrator user called `adminuser`. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.

For more information about IAM, see the following:

- AWS Identity and Access Management (IAM)
- Getting Started
- IAM User Guide

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 7)

Step 2: Set Up the AWS Command Line Interface (AWS CLI)

You use the AWS CLI to make interactive calls to Amazon Translate.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:
   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface
2. In the AWS CLI `config` file, add a named profile for the administrator user:

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   You use this profile when executing AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*. For a list of AWS Regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.
3. Verify the setup by typing the following help command at the command prompt:

   ```
   aws translate help
   ```

   You should see a brief description of Amazon Translate and a list of the available commands.

Next Step

Step 3: Getting Started (Console) (p. 8)

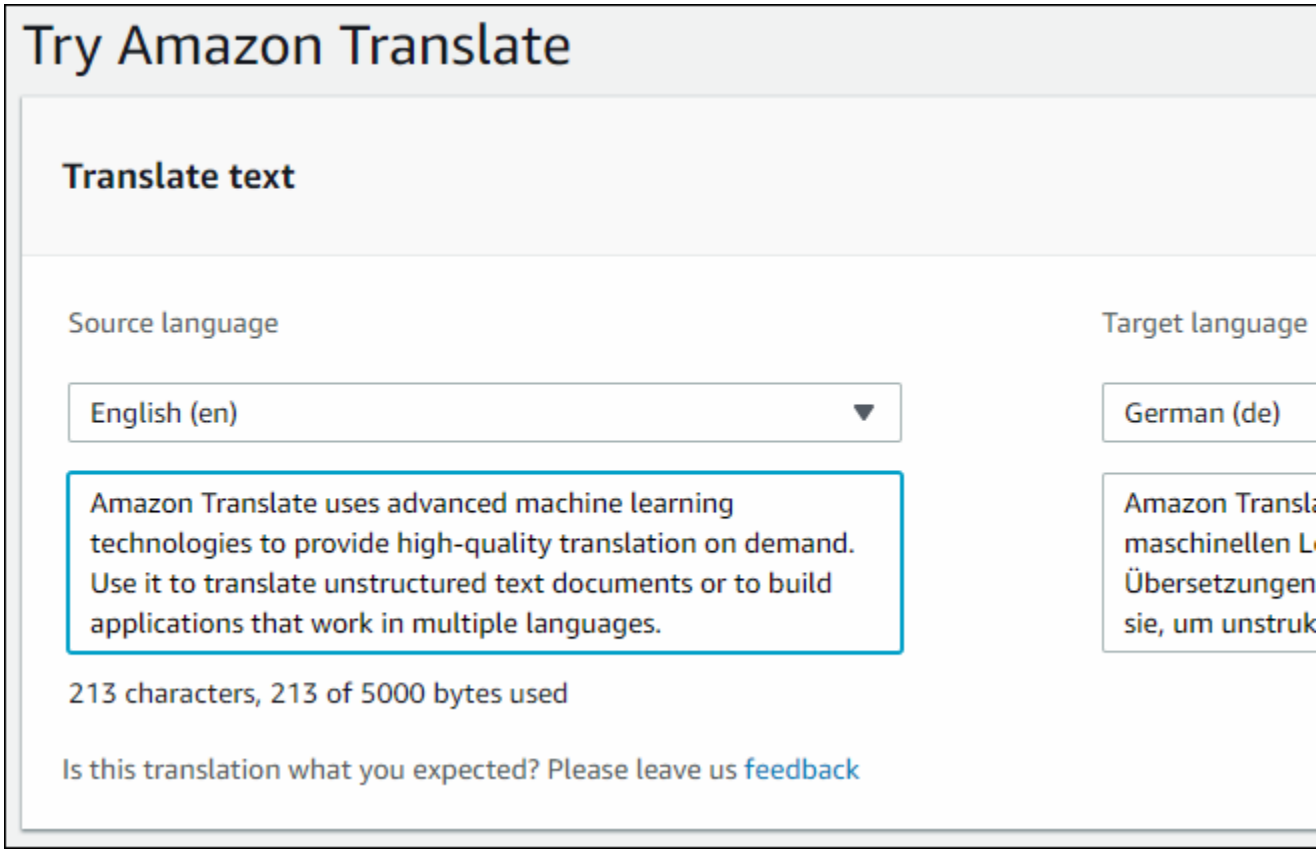
Step 3: Getting Started (Console)

The easiest way to get started with Amazon Translate is to use the console to translate some text. You can translate up to 5,000 characters using the console. If you haven't reviewed the concepts and terminology in How Amazon Translate Works (p. 4), we recommend that you do so before proceeding.

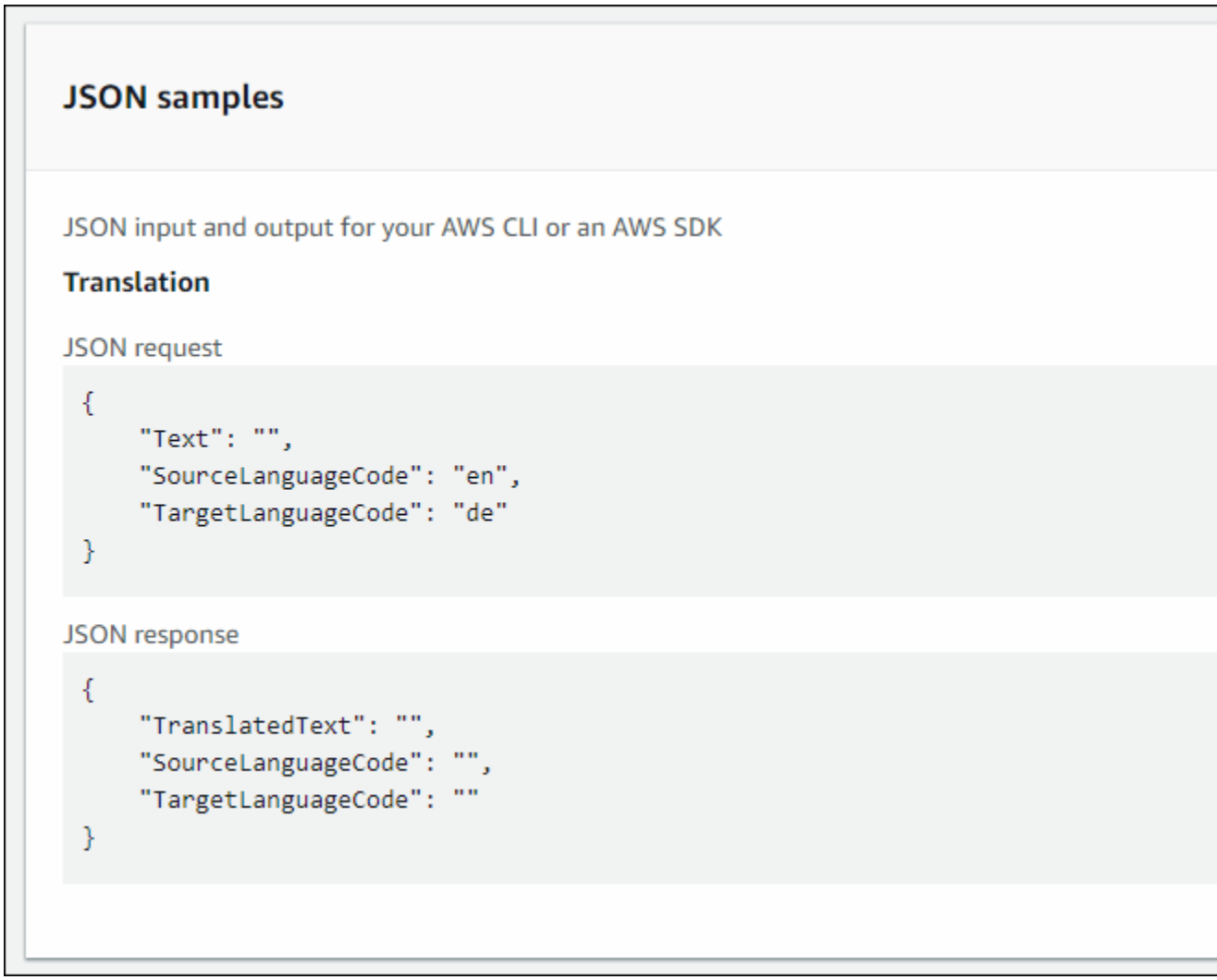
To start translating text, go to the AWS Management Console and open the Amazon Translate console.

If this is the first time that you've used Amazon Translate, choose **Launch real-time translation**.

In **Real-time translation**, choose the source and target languages. Enter the text that you want to translate in the left-hand text box. The translated text appears in the right-hand text box.



In the **JSON samples** section you can see the JSON input and output to the TranslateText (p. 83) operation.



Next Step

Step 4: Getting Started (AWS CLI) (p. 9)

Step 4: Getting Started (AWS CLI)

In the following exercises , you use the AWS command line interface (AWS CLI) to translate text. To complete these exercises, you need to be familiar with the CLI and have a text editor. For more information, see Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 7).

There are two ways to use the CLI to translate text with Amazon Translate. For short text, you can provide the text that you want to translate as a parameter of the `translate-text` command. For longer text, you can provide the source language, target language, and text in a JSON file.

To use Amazon Translate from the command line, you need to know the endpoint and region for the service. For a list of available endpoints and regions, see Amazon Translate Regions and Endpoints in the *AWS General Reference*.

Translate Text Using the Command Line

The following example shows how to use the `translate-text` operation from the command line to translate text. The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^). At the command line, type the following.

```
aws translate translate-text \
            --region region \
            --source-language-code "en" \
            --target-language-code "es" \
            --text "hello, world "
```

The response is the following JSON:

```
{
    "TargetLanguageCode": "es",
    "Text": "Hola, mundo",
    "SourceLanguageCode": "en"
}
```

Translate Text Using a JSON File

This example shows how to use the `translate-text` operation to translate a longer text block from a JSON file. You can specify the source and target language on the command line, but in this example, you specify them in the JSON file.

> **Note**
> The JSON file is formatted for readability. Reformat the `"Text"` field to remove line breaks. The example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

To translate text using a JSON file

1. Copy the following text into a JSON file called `translate.json`:

   ```
   {
       "Text": "Amazon Translate translates documents between languages in
       real time. It uses advanced machine learning technologies
       to provide high-quality real-time translation. Use it to
       translate documents or to build applications that work in
       multiple languages.",
       "SourceLanguageCode": "en",
       "TargetLanguageCode": "fr"
   }
   ```

2. In the AWS CLI, run the following command:

   ```
   aws translate translate-text \
               --region region \
               --cli-input-json file://translate.json > translated.json
   ```

   The command outputs a JSON file that contains the following JSON text:

   ```
   {
       "TargetLanguageCode": "fr",
       "Text": "Amazon Translate traduit les documents entre
   ```

```
    les langue en temps réel. Il utilise des technologies
    avancées d'apprentissage de la machine pour fournir
    une traduction en temps réel de haute qualité. Utilisez-le
    pour traduire des documents ou pour créer des applications
    qui fonctionnent en plusieurs langues.",
    "SourceLanguageCode": "en"
}
```

Next Step

To see other ways to use Amazon Translate see Examples (p. 22).

Step 5: Getting Started (SDK)

The following examples demonstrate how to use Amazon Translate TranslateText (p. 83) operation using Java and Python. Use them to learn about the `TranslateText` operation and as building blocks for your own applications.

To run the Java examples, you need to install the AWS SDK for Java. For instructions for installing the SDK for Java, see Set up the AWS SDK for Java.

Topics

Translating Text Using the AWS SDK for Java

The following example demonstrates using the TranslateText (p. 83) operation in Java. To run this example, you need the AWS SDK for Java. For instructions for installing the SDK for Java, see Set up the AWS SDK for Java.

```
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.translate.AmazonTranslate;
import com.amazonaws.services.translate.AmazonTranslateClient;
import com.amazonaws.services.translate.model.TranslateTextRequest;
import com.amazonaws.services.translate.model.TranslateTextResult;

public class App {
    private static final String REGION = "region";

    public static void main( String[] args ) {

        // Create credentials using a provider chain. For more information, see
        // https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/credentials.html
        AWSCredentialsProvider awsCreds = DefaultAWSCredentialsProviderChain.getInstance();

        AmazonTranslate translate = AmazonTranslateClient.builder()
                .withCredentials(new
 AWSStaticCredentialsProvider(awsCreds.getCredentials()))
                .withRegion(REGION)
```

```
                .build();

        TranslateTextRequest request = new TranslateTextRequest()
                .withText("Hello, world")
                .withSourceLanguageCode("en")
                .withTargetLanguageCode("es");
        TranslateTextResult result  = translate.translateText(request);
        System.out.println(result.getTranslatedText());
    }
}
```

You can change the source and target languages as long as the language pair selected is supported by Amazon Translate in the correct combination. For more information, see Supported Languages and Language Codes (p. 1)

Translating Text Using the AWS SDK for Python (Boto)

The following example demonstrates using the TranslateText (p. 83) operation in Python. To run it, you must first install Amazon Translate via the AWS CLI. For instructions, see the section called "Step 2: Set Up the AWS CLI" (p. 7).

```
import boto3

translate = boto3.client(service_name='translate', region_name='region', use_ssl=True)

result = translate.translate_text(Text="Hello, World",
            SourceLanguageCode="en", TargetLanguageCode="de")
print('TranslatedText: ' + result.get('TranslatedText'))
print('SourceLanguageCode: ' + result.get('SourceLanguageCode'))
print('TargetLanguageCode: ' + result.get('TargetLanguageCode'))
```

You can change the source and target languages as long as the language pair selected is supported by Amazon Translate in the correct combination. For more information, see Supported Languages and Language Codes (p. 1)

Custom Terminology

Another example, this one demonstrating using the Custom Terminology operations in Python:

```
    #!/usr/bin/env python
    # -*- coding: utf-8 -*-

    import boto3

    translate = boto3.client(service_name='translate')

    # The terminology file 'my-first-terminology.csv' has the following contents:
    '''
    en,fr
    Amazon Family,Amazon Famille
    '''

    # Read the terminology from a local file
    with open('/tmp/my-first-terminology.csv', 'rb') as f:
        data = f.read()

    file_data = bytearray(data)

    print("Importing the terminology into Amazon Translate...")
```

```
    response = translate.import_terminology(Name='my-first-terminology',
 MergeStrategy='OVERWRITE', TerminologyData={"File": file_data, "Format": 'CSV'})
    print("Terminology imported: "),
    print(response.get('TerminologyProperties'))
    print("\n")

    print("Getting the imported terminology...")
    response = translate.get_terminology(Name='my-first-terminology',
 TerminologyDataFormat='CSV')
    print("Received terminology: "),
    print(response.get('TerminologyProperties'))
    print("The terminology data file can be downloaded here: " +
 response.get('TerminologyDataLocation').get('Location'))
    print("\n")

    print("Listing the first 10 terminologies for the account...")
    response = translate.list_terminologies(MaxResults=10)
    print("Received terminologies: "),
    print(response.get('TerminologyPropertiesList'))
    print("\n")

    print("Translating 'Amazon Family' from English to French with no terminology...")
    response = translate.translate_text(Text="Amazon Family", SourceLanguageCode="en",
 TargetLanguageCode="fr")
    print("Translated text: " + response.get('TranslatedText'))
    print("\n")

    print("Translating 'Amazon Family' from English to French with the 'my-first-
terminology' terminology...")
    response = translate.translate_text(Text="Amazon Family", TerminologyNames=["my-first-
terminology"], SourceLanguageCode="en", TargetLanguageCode="fr")
    print("Translated text: " + response.get('TranslatedText'))
    print("\n")

    # The terminology file 'my-updated-terminology.csv' has the following contents:
    '''
    en,fr
    Amazon Family,Amazon Famille
    Prime Video, Prime Video
    '''

    # Read the terminology from a local file
    with open('/tmp/my-updated-terminology.csv', 'rb') as f:
        data = f.read()

    file_data = bytearray(data)

    print("Updating the imported terminology in Amazon Translate...")
    response = translate.import_terminology(Name='my-first-terminology',
 MergeStrategy='OVERWRITE', TerminologyData={"File": file_data, "Format": 'CSV'})
    print("Terminology updated: "),
    print(response.get('TerminologyProperties'))
    print("\n")

    print("Translating 'Prime Video' from English to French with no terminology...")
    response = translate.translate_text(Text="Prime Video", SourceLanguageCode="en",
 TargetLanguageCode="fr")
    print("Translated text: " + response.get('TranslatedText'))
    print("\n")

    print("Translating 'Prime Video' from English to French with the 'my-first-terminology'
 terminology...")
    response = translate.translate_text(Text="Prime Video", TerminologyNames=["my-first-
terminology"], SourceLanguageCode="en", TargetLanguageCode="fr")
    print("Translated text: " + response.get('TranslatedText'))
    print("\n")
```

```
    print("Cleaning up by deleting 'my-first-terminology'...")
    translate.delete_terminology(Name="my-first-terminology")
    print("Terminology deleted.")
```

Translating Text Using the AWS Mobile SDK for Android

You can use Amazon Translate in an Android application to translate text.

To configure the example

1. Set up the AWS Mobile SDK for Android. For instructions, see Android: Setup Options for the SDK in the *AWS Mobile Developer Guide*
2. Create an IAM user with the minimum required permissions to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity and Access Management User Guide*. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51). After you create the user, download the credentials or record the access key and secret access key.
3. Create a new project with Android Studio.
4. Add the following to the dependencies section of your `build.gradle` file.

```
dependencies {
    implementation 'com.amazonaws:aws-android-sdk-translate:2.6.20'
}
```

5. Add the following permissions to the `AndroidManifest.xml` file.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

6. Copy the source code into your project
7. Change the access key and secret access key to the keys that you recorded in step one.

Code

Use the following code to create the example.

```
package com.amazonaws.amazontranslatetester;

import android.app.Activity;
import android.util.Log;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.translate.AmazonTranslateAsyncClient;
import com.amazonaws.services.translate.model.TranslateTextRequest;
import com.amazonaws.services.translate.model.TranslateTextResult;

public class MainActivity extends Activity {

    private static final String LOG_TAG = MainActivity.class.getSimpleName();

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        AWSCredentials awsCredentials = new AWSCredentials() {
            @Override
            public String getAWSAccessKeyId() {
                return "access key";
            }

            @Override
            public String getAWSSecretKey() {
                return "secret key";
            }
        };

        AmazonTranslateAsyncClient translateAsyncClient = new
 AmazonTranslateAsyncClient(awsCredentials);
        TranslateTextRequest translateTextRequest = new TranslateTextRequest()
                .withText("Hello, world")
                .withSourceLanguageCode("en")
                .withTargetLanguageCode("es");
        translateAsyncClient.translateTextAsync(translateTextRequest, new
 AsyncHandler<TranslateTextRequest, TranslateTextResult>() {
            @Override
            public void onError(Exception e) {
                Log.e(LOG_TAG, "Error occurred in translating the text: " +
 e.getLocalizedMessage());
            }

            @Override
            public void onSuccess(TranslateTextRequest request, TranslateTextResult
 translateTextResult) {
                Log.d(LOG_TAG, "Original Text: " + request.getText());
                Log.d(LOG_TAG, "Translated Text: " +
 translateTextResult.getTranslatedText());
            }
        });
    }
}
```

Translating Text Using the AWS Mobile SDK for iOS

You can use Amazon Translate in an iOS application to translate text.

To configure the example

1. Create an IAM user with the minimum required permissions to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity and Access Management User Guide*. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51). After you create the user, download the credentials or record the access key and secret access key.
2. Install Xcode version 8.0 or later. You can download the latest version of Xcode from the Apple website, https://developer.apple.com/xcode/.
3. Install Cocoapods. In a terminal window, run the following command:

   ```
   sudo gem install cocoapods
   ```

4. Create a project using Xcode. Then, in a terminal window, navigate to the directory that contains your project's `.xcodeproj` file and run the following command:

   ```
   pod init
   ```

5. Add the core Mobile SDK for iOS components to your pod file:

```
platform :ios, '9.0'
target :'app name' do
   use_frameworks!
   pod 'AWSTranslate', '~> 2.6.19'
   # other pods
end
```

6. Install dependencies by running the following command in a terminal window:

```
pod install --repo-update
```

7. Running '`pod install`' creates a new workspace file. Close your Xcode project and then open it using the ./*project_name*.xcworkspace file. From now on you should only use this file to open your Xcode project

   Rebuild your app after you open it to resolve APIs from the new libraries called in your code.

8. Add the following import statement to your view controller:

```
import AWSTranslate
```

9. Copy the following code into your XCode project. Update the access key and secret key to the values that you recorded in step 1.

Code

Use the following code to create the example.

```
var credentialsProvider = AWSStaticCredentialsProvider(accessKey: "access key", secretKey:
 "secret key")

var configuration = AWSServiceConfiguration(region: AWSRegionUSEast1, credentialsProvider:
 credentialsProvider)

AWSServiceManager.default().defaultServiceConfiguration = configuration

let translateClient = AWSTranslate.default()
let translateRequest = AWSTranslateTranslateTextRequest()
translateRequest?.sourceLanguageCode = "en"
translateRequest?.targetLanguageCode = "es"
translateRequest?.text = "Hello World"

let callback: (AWSTranslateTranslateTextResponse?, Error?) -> Void = { (response, error) in
   guard let response = response else {
      print("Got error \(error)")
      return
   }

   if let translatedText = response.translatedText {
      print(translatedText)
   }
}

translateClient.translateText(translateRequest!, completionHandler: callback)
```

Custom Terminology

Using custom terminology with your translation requests enables you to make sure that your brand names, character names, model names, and other unique content is translated exactly the way you need it, regardless of its context and the Amazon Translate algorithm's decision.

It's easy to set up a terminology file and attach it to your Amazon Translate account. When you translate text, you simply choose to use the custom terminology as well, and any examples of your source word are translated as you want them.

For example, consider the following: *Amazon Family* is a collection of benefits that offers Amazon Prime members exclusive offers, such as up to 20% off subscriptions to diapers, baby food, and more. In France, it's called *Amazon Famille*. If you translate *Amazon Family* into French using Amazon Translate without any additional context, the result is *Famille Amazon* as the output. While this is an accurate translation, it isn't what the Amazon team in France needs. However, if you add context to this, as in: "Have you ever shopped with Amazon Family?", Amazon Translate determines that the program name does not need to be translated, and the result is "Avez-vous déjà fait des achats avec Amazon Family?". This is a good translation, but still not what the Amazon team is looking for. Custom terminologies can solve issues like this. By adding an entry showing that the term *Amazon Family* should be translated as *Amazon Famille* to their custom terminology, the team can make sure that it is translated this way every time, regardless of context. *Amazon Family* is now translated into *Amazon Famille* and "Have you ever shopped with Amazon Family?" is now translated into "Avez-vous déjà fait des achats avec Amazon Famille?"

Topics

How does this work?

Generally speaking, when a translation request comes in, Amazon Translate reads the source sentence, creates a semantic representation of the content (in other words, it understands it), and generates a translation into the target language.

When a custom terminology is used as part of the translation request, the engine scans the terminology file before returning the final result. When the engine identifies an exact match between a terminology entry and a string in the source text, it locates the appropriate string in the proposed translation and replaces it with the terminology entry. In the Amazon Family example, it first generates the translation "Avez-vous déjà fait des achats avec Amazon Family?" but stops and replaces *Amazon Family* with *Amazon Famille* before providing the response.

Creating a Custom Terminology

You can use either a CSV file or a TMX file with the source text and the target (translated) term for the terminology file. A single source text is used for each term, but there can be multiple target terms, one for each language, as long as the target and source language can be used.

CSV (comma separated values)

The first column contains the source text, and the other columns contain the target translations. The first row consists of the language codes, with the source language in the first columns, and target language translations in the others.

en	fr	de	es	
Amazon	Amazon	Amazon	Amazon	

TMX (Translation Memory eXchange)

A TMX file is an XML-type file commonly used by translation software. Although the form is different than the CSV, the content is similar:

```
<?xml version="1.0" encoding="UTF-8"?>
<tmx version="1.4">
 <header
    creationtool="XYZTool" creationtoolversion="0"
    datatype="PlainText" segtype="sentence"
    adminlang="en-us" srclang="en"
    o-tmf="test"/>
 <body>
   <tu>
     <tuv xml:lang="en">
       <seg>Amazon</seg>
     </tuv>
     <tuv xml:lang="fr">
       <seg>Amazon</seg>
     </tuv>
     <tuv xml:lang="de">
       <seg>Amazon</seg>
     </tuv>
     <tuv xml:lang="es">
       <seg>Amazon</seg>
     </tuv>
   </tu>
 </body>
</tmx>
```

These files are then attached to your Amazon Translate account. When a translation job is run and you choose to use the custom terminology, Amazon Translate then uses the designated word whenever it encounters the source word.

Important
The source word within a custom terminology is *case-sensitive* and will not work for words that are not an exact match.

Compatible Languages

Some languages do not change the shape of a word based on sentence context. With these languages, applying a custom terminology is most likely to improve overall translation quality. However, some languages do have extensive word shape changes. We do not recommend applying the feature to those languages, but we do not restrict you from doing so.

The following table shows you languages and recommendations for using this feature:

Language	Recommended/Not Recommended
East Asian Languages (Chinese, Indonesian, Japanese, Korean, Malay)	Recommended

Language	Recommended/Not Recommended
Germanic Languages (Danish, Dutch, English, German, Norwegian, Swedish)	Recommended
Romance Languages (French, Italian, Romanian, Spanish, Portuguese)	Recommended
Semitic Languages (Arabic, Hebrew)	Recommended
Indo-Iranian (Hindi, Persian, Urdu)	Recommended
Greek	Recommended
Thai	Recommended
Vietnamese	Recommended
Balto-Slavic Languages (Czech, Polish, Russian, Ukrainian)	Not recommended
Uralic Languages (Finnish, Hungarian)	Not recommended
Arabic	Not recommended
Turkish	Not recommended

Using Custom Terminologies

When using a Custom Terminology when translating text using the TranslateText (p. 83) operation, the process is similar to when you translate text without one. The difference is that when you call the operation with the Custom Terminology, you also include the optional `TerminologyNames` parameter.

For example, if you have the following terminology file called `Amazon_Family.csv` attached to your account:

```
en,fr
Amazon Family,Amazon Famille
```

You can use the following to call the Custom Terminology to translate your text using the CLI:

Note
This example is formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

```
aws translate translate-text \
     --region region \
     --source-language-code "en" \
     --target-language-code "fr" \
     --terminology-names "Amazon_Family" \
     --text "Have you ever shopped with Amazon Family?"
```

This uses the selected Custom Terminology to translate this text as "Avez-vous déjà fait des achats avec Amazon Famille?" instead of the direct (but undesirable) translation "Avez-vous déjà fait des achats avec Famille Amazon?"

Using Python, the same task can be seen with the following:

```
import boto3

translate = boto3.client(service_name='translate')

print("Translating 'Have you ever shopped with Amazon Family?' from English to French with
 the 'Amazon_Family' custom terminology...")
response = translate.translate_text(Text="Have you ever shopped with Amazon Family?",
 TerminologyNames=["Amazon_Family"], SourceLanguageCode="en", TargetLanguageCode="fr")
print("Translated text: " + response.get('TranslatedText'))
print("\n")
```

For more information on using the Amazon Translate operations with Custom Terminologies, see Actions (p. 71).

Encrypting Your Terminology

Amazon Translate endeavors to protect all of your data and your custom terminologies are no different. When created, each custom terminology is encrypted so it accessible only by you.

Three encryption options are available:

- Using AWS encryption. This is the default option and is used to safeguard your information if you don't choose another method.
- Using an encryption key associated with your account. If this is chosen, a menu in the console provides you with a choice of associated encryption keys to use.
- Using an encryption key not associated with your account. If you choose this option, a box is displayed in the console for you to enter the Amazon Resource Name (ARN) of the encryption key.

Best Practices

The following are suggested best practices when using custom terminologies

- Keep your custom terminology minimal. Only include words which you want to control and which are completely unambiguous. Only use words that you know you will never want to use an alternate meaning of, and you want it to only ever be translated in a single way. Ideally, limit the list to proper names, like brand names and product names.
- Custom terminologies are case-sensitive. If you need both capitalized and non-capitalized versions of a word to be included, you must include an entry for each version.
- Do not include different translations for the same source phrase (for example, entry #1-EN: Amazon, FR: Amazon; entry #2-EN: Amazon FR: Amazone).
- Some languages do not change the shape of a word based on sentence context. With these languages, applying a custom terminology is most likely to improve overall translation quality. However, some languages do have extensive word shape changes. We do not recommend applying the feature to those languages, but we do not restrict you from doing so. For more information, see Compatible Languages (p. 19).

Examples

The following examples show ways that you can use Amazon Translate.

Topics

Using Amazon Polly with Amazon Translate

To speak translated text, you can use Amazon Polly with Amazon Translate. In this example you'll create a Web page where you can translate text using Amazon Translate and then speak that text using Amazon Polly. The code can be summarized into the following:

- CSS and HTML to create the Web page.
- Initialization code that creates controllers for Amazon Translate and Amazon Polly.
- A function that reads data from the Web page and calls Amazon Translate.
- A function that reads data from the Web page and calls Amazon Polly.
- Utility functions for managing the Web page.

To configure the example

1. Install and Configure the AWS SDK for JavaScript. For instructions for installing the SDK for JavaScript, see Installing the SDK for JavaScript.
2. Copy the code for the example to an HTML file on your Web server.
3. Update the `<script>` tag to the location where you installed the SDK for JavaScript.
4. Change the region and endpoint to the region where you want to run the Amazon Translate and Amazon Polly operations. For a list of supported regions for Amazon Translate, see AWS Regions and Endpoints. For a list of supported regions for Amazon Polly, see AWS Regions and Endpoints in the *AWS General Reference*.
5. Create an IAM user with the minimum required permissions to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity and Access Management User Guide*. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51) and Using Identity-Based Policies (IAM Policies) for Amazon Polly in the *Amazon Polly Developer Guide*.
6. Provide the access ID and secret key of the IAM user created in the previous step.

Code

The following is the complete code of the example Web page. You can copy this code into an HTML file to run the example on your own Web server.

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Amazon Translate</title>
    <script src="aws-sdk/dist/aws-sdk.js"></script>
</head>

<body>
    <h1 style="text-align: left">Amazon Translate Demo</h1>
    <br/>
    <table class="tg">
        <tr>
            <th align="left">
Source Language Code:
                <select id="sourceLanguageCodeDropdown">
                      <option value="en">en</option>
                      <option value="ar">ar</option>
                      <option value="cs">cs</option>
                      <option value="de">de</option>
                      <option value="es">es</option>
                      <option value="fr">fr</option>
                      <option value="it">it</option>
                      <option value="ja">ja</option>
                      <option value="pt">pt</option>
                      <option value="ru">ru</option>
                      <option value="tr">tr</option>
                      <option value="zh">zh</option>
                      <option value="zh-TW">zh-TW</option>
                </select>
            </th>
            <th align="left">
Target Language Code:
                <select id="targetLanguageCodeDropdown">
                      <option value="en">en</option>
                      <option value="ar">ar</option>
                      <option value="cs">cs</option>
                      <option value="de">de</option>
                      <option value="es">es</option>
                      <option value="fr">fr</option>
                      <option value="it">it</option>
                      <option value="ja">ja</option>
                      <option value="pt">pt</option>
                      <option value="ru">ru</option>
                      <option value="tr">tr</option>
                      <option value="zh">zh</option>
                      <option value="zh-TW">zh-TW</option>
                </select>
            </th>
        </tr>
        <tr>
            <th>
                <textarea id="inputText" name="inputText" rows="10" cols="50"
 placeholder="Text to translate..."></textarea>
            </th>
            <th>
                <textarea id="outputText" name="outputText" rows="10" cols="50"
 placeholder="Translated text..."></textarea>
            </th>
        </tr>
        <tr>
            <th align="left">
                <button type="button" name="translateButton"
 onclick="doTranslate()">Translate</button>
                <button type="button" name="synthesizeButton"
 onclick="doSynthesizeInput()">Synthesize Input Speech</button>
                <button type="button" name="clearButton" onclick="clearInputs()">Clear</
button>
```

```
            </th>
            <th align="left">
                <button type="button" name="synthesizeButton"
 onclick="doSynthesizeOutput()">Synthesize Output Speech</button>
            </th>
        </tr>
    </table>
    <script type="text/javascript">
        // set the focus to the input box
        document.getElementById("inputText").focus();

        /**
        * Change the region and endpoint.
        */
        AWS.config.region = 'region'; // Region

         /**
         * In a production application you should use a secure method of authenticating
 uses, such as the ones
         * described here:
         *   https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-
credentials-browser.html
         *
         * Note that Amazon Translate does not work with Amazon Cognito Identity.
         *
         * For this example you place the credentials of an IAM user in the HTML page. The
 IAM user associated
         * with these credentials must have permissions to call Amazon Translate. We
 recommend using the following
         * permissions policy and nothing more, as anyone that has access to this HTML page
 will also have access to
         * these hard-coded credentials.
         * {
         *     "Version": "2012-10-17",
         *     "Statement": [
         *         {
         *             "Action": [
         *                 "translate:TranslateText",
         *                 "polly:SynthesizeSpeech"
         *             ],
         *             "Resource": "*",
         *             "Effect": "Allow"
         *         }
         *     ]
         * }
         *
         * For more information about the AWS Credentials object, see:
         *   http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/Credentials.html
         */
        AWS.config.credentials = new AWS.Credentials("access key", "secret key");

        var translate = new AWS.Translate({region: AWS.config.region});
        var polly = new AWS.Polly();

        function doTranslate() {
            var inputText = document.getElementById('inputText').value;
            if (!inputText) {
                alert("Input text cannot be empty.");
                exit();
            }

            // get the language codes
            var sourceDropdown = document.getElementById("sourceLanguageCodeDropdown");
            var sourceLanguageCode =
 sourceDropdown.options[sourceDropdown.selectedIndex].text;
```

```
            var targetDropdown = document.getElementById("targetLanguageCodeDropdown");
            var targetLanguageCode =
targetDropdown.options[targetDropdown.selectedIndex].text;

            var params = {
                Text: inputText,
                SourceLanguageCode: sourceLanguageCode,
                TargetLanguageCode: targetLanguageCode
            };

            translate.translateText(params, function(err, data) {
                if (err) {
                    console.log(err, err.stack);
                    alert("Error calling Amazon Translate. " + err.message);
                    return;
                }
                if (data) {
                    var outputTextArea = document.getElementById('outputText');
                    outputTextArea.value = data.TranslatedText;
                }
            });
        }

        function doSynthesizeInput() {
            var text = document.getElementById('inputText').value.trim();
            if (!text) {
                return;
            }
            var sourceLanguageCode =
document.getElementById("sourceLanguageCodeDropdown").value;
            doSynthesize(text, sourceLanguageCode);
        }

        function doSynthesizeOutput() {
            var text = document.getElementById('outputText').value.trim();
            if (!text) {
                return;
            }
            var targetLanguageCode =
document.getElementById("targetLanguageCodeDropdown").value;
            doSynthesize(text, targetLanguageCode);
        }

        function doSynthesize(text, languageCode) {
            var voiceId;
            switch (languageCode) {
                case "de":
                    voiceId = "Marlene";
                    break;
                case "en":
                    voiceId = "Joanna";
                    break;
                case "es":
                    voiceId = "Penelope";
                    break;
                case "fr":
                    voiceId = "Celine";
                    break;
                case "pt":
                    voiceId = "Vitoria";
                    break;
                default:
                    voiceId = null;
                    break;
            }
            if (!voiceId) {
```

```
                alert("Speech synthesis unsupported for language code: \"" + languageCode +
 "\"");
                return;
            }
            var params = {
                OutputFormat: "mp3",
                SampleRate: "8000",
                Text: text,
                TextType: "text",
                VoiceId: voiceId
            };
            polly.synthesizeSpeech(params, function(err, data) {
                if (err) {
                    console.log(err, err.stack); // an error occurred
                    alert("Error calling Amazon Polly. " + err.message);
                }
                else {
                    var uInt8Array = new Uint8Array(data.AudioStream);
                    var arrayBuffer = uInt8Array.buffer;
                    var blob = new Blob([arrayBuffer]);
                    var url = URL.createObjectURL(blob);

                    audioElement = new Audio([url]);
                    audioElement.play();
                }
            });
        }

        function clearInputs() {
            document.getElementById('inputText').value = "";
            document.getElementById('outputText').value = "";
            document.getElementById("sourceLanguageCodeDropdown").value = "en";
            document.getElementById("targetLanguageCodeDropdown").value = "en";
        }
    </script>
</body>

</html>
```

Using Amazon Translate to Translate a Chat Channel

You can use Amazon Translate for real time translation of chat messages. This example uses a Twitch channel, but you can use it as a starting point for other real-time streaming text like other chat platforms, customer service interactions, message boards, and more.

This example uses a web page that shows real-time messages in English and their real-time translations side-by-side. You can send the messages to Amazon Polly to speak the text. To follow a person in the chat, type their user name. The app will speak only messages from that user.

The code can be summarized as follows:

- CSS and HTML to create the Web page.
- Initialization code that creates controllers for Amazon Translate and Amazon Polly.
- A call back function that gets executed when a chat message is received.
- A function that sends a chat message.
- A function that calls Amazon Translate to translate messages.

- A function that calls Amazon Polly to synthesize speech.
- Utility functions for managing the Web page.

To configure the example

1. Install and Configure the AWS SDK for JavaScript. For instructions for installing the SDK for JavaScript, see Installing the SDK for JavaScript.
2. Copy the code for the example to an HTML file on your Web server.
3. Update the `<script>` tag to the location where you installed the SDK for JavaScript.
4. Change the region and endpoint to the region where you want to run the Amazon Translate and Amazon Polly operations. For a list of supported regions for Amazon Translate, see AWS Regions and Endpoints in the *AWS General Reference*.
5. Create an IAM user with the minimum required permissions to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity and Access Management User Guide*. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51) and Using Identity-Based Policies (IAM Policies) for Amazon Polly in the *Amazon Polly Developer Guide*.
6. Provide the access ID and secret key of the IAM user created in the previous step.
7. Provide a Twitch user name and OAuth token for your account. You can create a Twitch account at https://www.twitch.tv. You can create a Twitch OAuth token at https://twitchapps.com/tmi.

```
<!doctype html>
<html lang="en">
<head>
  <title>Amazon Translate</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Latest compiled and minified CSS for Bootstrap -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
K68vbdEjh4u" crossorigin="anonymous">

   <!-- Custom CSS -->
   <style>
     .topHeader
     {
       background-color: #6441a4;
       padding: 10px;
       border-bottom: solid 1px #cacaca;
       color: white
     }

     .panelHeading
     {
       background-color: #6441a4 !important;
     }

     .panelBody
     {
      min-height: 450px;  max-height: 450px;overflow-y: scroll;
     }

     body{
        margin-left: 0px;
        margin-right: 0px;
        height: 100%;
      }
```

```
   </style>
</head>
<body>
  <div class="container-fluid">
    <!--Top Header-->
    <div class="row topHeader">
      <div class="col-md-12">
          <h4>Amazon Translate - Artificial Intelligence on AWS - Powerful machine learning
 for all Developers and Data Scientists</h4>
      </div>
    </div>

    <!--Status Label-->
    <div class="row">
      <div class="col-md-12">
        <p class="bg-info">
        <div id="connecting-div"></div>
        </p>
      </div>
    </div>

    <div class="row" style="padding: 10px;">
      <div class="col-md-6">
          <div class="form-inline">
            <div class="form-group">
              <input type="text" id="channel" class="form-control" value=""
 placeholder="Channel"/>
            </div>
            <div class="form-group">
              <select id="sourceLanguage" class="form-control">
                  <option value="en">en</option>
                  <option value="ar">ar</option>
                  <option value="de" selected="selected">de</option>
                  <option value="es">es</option>
                  <option value="fr">fr</option>
                  <option value="pt">pt</option>
                  <option value="zh">zh</option>
                </select>
            </div>
            <div class="form-group">
              <select id="targetLanguage" class="form-control">
                <option value="en" selected="selected">en</option>
                <option value="ar">ar</option>
                <option value="de">de</option>
                <option value="es">es</option>
                <option value="fr">fr</option>
                <option value="pt">pt</option>
                <option value="zh">zh</option>
              </select>
            </div>
            <div class="form-group">
              <button type="button" class="form-control" id="btn-go"
 onclick="connect()">Go</button>
              <button type="button" class="form-control" id="btn-stop"
 onclick="location.href='index.html';">Stop</button>
              <span id="status"></span>
            </div>
          </div>
        </div>
        <div class="col-md-6">
            <div class="form-inline">
              <div class="form-group">
                  <input type="checkbox" id="cbSpeak" value="Speak"> Speak Live Translation
                  <input type="text" id="follow" class="form-control" value=""
 placeholder="follow"/>
                </div>
```

```
            </div>
          </div>
      </div>

    <!--Chat Boxes-->
    <div class="row">
      <!--Live Chat-->
      <div class="col-md-6">
        <div class="panel panel-primary">
          <div class="panel-heading panelHeading">Live Chat</div>
            <div id="livechatc" class="panel-body panelBody">
              <div class="subscribe" id="livechat"></div>
          </div>
        </div>
      </div>
      <!--Live Chat-->
      <!--Translated Chat-->
      <div class="col-md-6">
        <div class="panel panel-primary">
          <div class="panel-heading panelHeading">Live Translation</div>
            <div id="livetranslationc" class="panel-body panelBody">
              <div class="imageDetected" id="livetranslation"></div>
          </div>
        </div>
      </div>
      <!--Translated Chat-->
    </div>

    <!--Send Message-->
    <div class="row">
        <div class="col-md-11">
            <input type="text" id="message" class="form-control"/>
        </div>
        <div class=" col-md-1">
          <button type="button" class="form-control btn btn-default" id="btn-send"
 onclick="sendMessage()">Send</button>
        </div>
    </div>
  </div>

    <!-- Latest compiled and minified JavaScript -->
    <!-- jQuery first, then Bootstrap JS -->
    <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://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
 integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
 crossorigin="anonymous"></script>

    <script src="aws-js-sdk/dist/aws-sdk-all.js"></script>
    <script src="http://cdn.tmijs.org/js/1.2.1/tmi.min.js" integrity="sha384-
eE0n7sm1W7DOUI2Xh5I4qSpZTe6hupAO0ovLfqEy0yVJtGRBNfssdmjbJhEYm6Bw"
 crossorigin="anonymous"></script>
    <script>
      cred = {
           twitchUsername: "Twitch user name",
           twitchOAuthToken: "Twitch OAuth token",
           awsAccessKeyId: "access key",
           awsSecretAccessKey: "secret key"
      };

      AWS.config.region = 'region';
      ep = new AWS.Endpoint('endpoint');

      AWS.config.credentials = new AWS.Credentials(cred.awsAccessKeyId,
 cred.awsSecretAccessKey);
```

```
window.translator = new AWS.Translate({endpoint: ep, region: AWS.config.region});

/*************************Init and Connect to Chat***************************/
function connect(){
  init();

  //Twitch Client
  var options = {
      options: {
        debug: false
      },
      connection: {
        cluster: "aws",
        reconnect: true
      },
      identity: {
        username: cred.twitchUsername,
        password: cred.twitchOAuthToken
      },
      channels: [con.channel]
  };

  window.client = tmi.client(options);

  window.client.connect();

  //Attached Handlers
  window.client.on("chat", onChat);
  window.client.on("connecting", onConnecting);
  window.client.on("connected", onConnected);

  //Disable UI Elements
  document.getElementById("sourceLanguage").disabled = true;
  document.getElementById("targetLanguage").disabled = true;
  document.getElementById("channel").disabled = true;
  document.getElementById("btn-go").disabled = true;
}

function init(){
  //Get UI Controls
  var lc = document.getElementById("livechat");
  var lt = document.getElementById("livetranslation")
  var lcc = document.getElementById("livechatc");
  var ltc = document.getElementById("livetranslationc")
  var cbspeak = document.getElementById("cbSpeak")
  var follow = document.getElementById("follow");
  var sendMessage = document.getElementById("message");

  //Cache values
  con = {
    channel: document.getElementById("channel").value,
    sourceLanguage: document.getElementById("sourceLanguage").value,
    targetLanguage: document.getElementById("targetLanguage").value,
    liveChatUI: lc,
    liveTranslationUI: lt,
    liveChatUIContainer: lcc,
    liveTranslationUIContainer: ltc,
    cbSpeak: cbspeak,
    follow: follow,
    sendMessage: sendMessage
  }

    lc.innerHTML = '';
    lt.innerHTML = '';

    //Speaker
```

```
        var voiceId = "Joanna";
        if(con.targetLanguage == "en")
           voiceId = "Joanna";
        else if(con.targetLanguage == "de")
             voiceId = "Marlene";
        else if(con.targetLanguage == "es")
            voiceId = "Conchita";
        else if(con.targetLanguage == "fr")
           voiceId = "Celine";
        else if(con.targetLanguage == "pt")
           voiceId = "Ines";
        else
           voiceId = "Joanna";
         window.audioPlayer = AudioPlayer(voiceId);
    }
    /*************************Init and Connect to Chat*****************************/

    /*************************Receive and Translate Chat*****************************/
    function onChat (channel, userstate, message, self) {
        // Don't listen to my own messages..
        if (self) return;

        //Translate
        if (message) {
          var username = userstate['username'];

          var params = {
              Text: message,
              SourceLanguageCode: con.sourceLanguage,
              TargetLanguageCode:  con.targetLanguage
          };

          window.translator.translateText(params, function onIncomingMessageTranslate(err,
data) {
             if (err) {
                console.log("Error calling Translate. " + err.message + err.stack);
            }
             if (data) {
                 console.log("M: " + message);
                 console.log("T: " + data.TranslatedText);

                 //Print original message in chat UI
                 con.liveChatUI.innerHTML += '<strong>' + username + '</strong>: ' +
message + '<br>';

                 //Print translation in translation UI
                 con.liveTranslationUI.innerHTML += '<strong>' + username + '</strong>: '
+ data.TranslatedText + '<br>';

                 //If speak translation in enabled, speak translated message
                 if(con.cbSpeak.checked){
                   if(con.follow.value == "" || username == con.follow.value)
                     audioPlayer.Speak(username + " says " + data.TranslatedText);
                 }

                 //Scroll chat and translated UI to bottom to keep focus on latest
messages
                 con.liveChatUIContainer.scrollTop = con.liveChatUIContainer.scrollHeight;
                 con.liveTranslationUIContainer.scrollTop =
con.liveTranslationUIContainer.scrollHeight;
               }
          });
        }
    }
    /*************************Receive and Translate Chat*****************************/
```

```
    /*************************Client Connecting*****************************/
    function onConnecting (address, port) {
        document.getElementById("status").innerHTML = " [ Connecting...]"
    }

    function onConnected (address, port) {
        document.getElementById("status").innerHTML = " [ Connected ]"
        window.audioPlayer.Speak("Connected to channel " + con.channel + ". You should now
be getting live chat messages.");
    }
    /*************************Client Connecting*****************************/

    /*************************Send Message*****************************/
    function sendMessage(){
         if(con.sendMessage.value){
           message = con.sendMessage.value;
           var params = {
               Text: con.sendMessage.value,
               SourceLanguageCode: con.targetLanguage,
               TargetLanguageCode:  con.sourceLanguage
           };

           window.translator.translateText(params, function onSendMessageTranslate(err,
data) {
                 if (err) {
                    console.log("Error calling Translate. " + err.message + err.stack);
                 }
                 if (data) {
                     console.log("M: " + message);
                     console.log("T: " + data.TranslatedText);

                     //Send message to chat
                     window.client.action(con.channel, data.TranslatedText);

                     //Clear send message UI
                     con.sendMessage.value = "";

                     //Print original message in Translated UI
                     con.liveTranslationUI.innerHTML += '<strong> ME: </strong>: ' +
message + '<br>';

                     //Print translated message in Chat UI
                     con.liveChatUI.innerHTML += '<strong> ME: </strong>: ' +
data.TranslatedText + '<br>';

                     //Scroll chat and translated UI to bottom to keep focus on latest
messages
                     con.liveChatUIContainer.scrollTop =
con.liveChatUIContainer.scrollHeight;
                     con.liveTranslationUIContainer.scrollTop =
con.liveTranslationUIContainer.scrollHeight;
                 }
           });
         }
    }
    /*************************Send Message*****************************/

    /*************************Audio player*****************************/
    function AudioPlayer(voiceId) {
        var audioPlayer = document.createElement('audio');
        audioPlayer.setAttribute("id", "audioPlayer");
        document.body.appendChild(audioPlayer);

        var isSpeaking = false;

        var speaker = {
```

```
    self: this,
    playlist:[],

    Speak: function (text) {
        //If currently speaking a message, add new message to the playlist
        if (isSpeaking) {
            this.playlist.push(text);
        } else {
            speakTextMessage(text).then(speakNextTextMessage)
        }
    }
}

// Speak text message
function speakTextMessage(text) {
    return new Promise(function (resolve, reject) {
        isSpeaking = true;
        getAudioStream(text).then(playAudioStream).then(resolve);
    });
}

// Speak next message in the list
function speakNextTextMessage() {
    var pl = speaker.playlist;
    if (pl.length > 0) {
        var txt = pl[0];
        pl.splice(0, 1);
        speakTextMessage(txt).then(speakNextTextMessage);
    }
}

// Get synthesized speech from Amazon polly
function getAudioStream(textMessage) {
    return new Promise(function (resolve, reject) {
        var polly = new AWS.Polly();
        var params = {
            OutputFormat: 'mp3',
            Text: textMessage,
            VoiceId: voiceId
        }
        polly.synthesizeSpeech(params, function (err, data) {
            if (err)
                reject(err);
            else
                resolve(data.AudioStream);
        });
    });
}

// Play audio stream
function playAudioStream(audioStream) {
    return new Promise(function (resolve, reject) {
        var uInt8Array = new Uint8Array(audioStream);
        var arrayBuffer = uInt8Array.buffer;
        var blob = new Blob([arrayBuffer]);

        var url = URL.createObjectURL(blob);
        audioPlayer.src = url;
        audioPlayer.addEventListener("ended", function () {
            isSpeaking = false;
            resolve();
        });
        audioPlayer.play();
    });
}
```

```
        return speaker;
    }
    /*************************Audio player*****************************/
  </script>
</body>
</html>
```

Using Amazon Translate with Amazon DynamoDB

This example shows you how to translate a product review and store it in Amazon DynamoDB. If you request the same review later, DynamoDB returns it without Amazon Translate needing to translate it again.

In this example, you:

- Use AWS CloudFormation to create DynamoDB tables to store the translation and a Lambda function that calls the TranslateText (p. 83) operation.
- Test the function using the AWS Lambda console.

To run the example

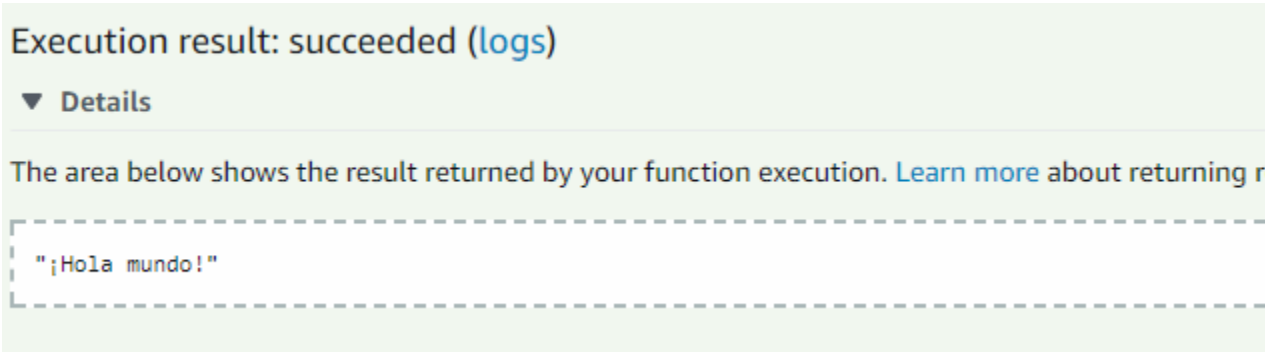
1. Copy the contents of `example.py`, which you can find in Python Lambda Function (p. 35), to a file named `example.py`. `example.py` is a Lambda function that calls the TranslateText (p. 83) operation. Compress the file to a zip archive named `example.zip`. Store it in an S3 bucket in the same AWS Region where you want to run the function.
2. Create a new file named `template.yaml`. Copy the AWS CloudFormation template code, which you can find in AWS CloudFormation Template (p. 36), into the file. AWS CloudFormation uses the template to create resources for the sample application. Change `BUCKET_NAME` to the name of the S3 bucket that contains `example.zip`. Save the file in a local directory.
3. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
4. Choose **Create new stack**.
5. Choose **Upload a template to Amazon S3**, and then choose **Choose file**. Choose `template.yaml`, that you created in Step 2, then **Next**.
6. Type a name for the stack, then choose **Next**.
7. On the **Options** page, choose **Next**.
8. Choose **I acknowledge that AWS CloudFormation might create IAM resources** and **I acknowledge that AWS CloudFormation might create IAM resources with custom names**. For more information, see Controlling Access with AWS Identity and Access Management in the *AWS CloudFormation User Guide*.
9. Choose **Create Change Set**.
10. After AWS CloudFormation creates the change set, choose **Execute**. Wait until AWS CloudFormation creates the stack.
11. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
12. Choose the new function. Its name starts with `TestTranslate-ReviewTranslate`.
13. On the function detail page, choose **Test**.
14. For **Event name**, type **`TestTranslate`**. For **Configure test event**, replace the JSON with the following:

```
{
```

```
   "review": "hello world",
   "target_language": "es",
   "source_language": "en",
   "review_id": "1"
}
```

Choose **Create**.

15. Make sure that **TestTranslate** is selected, then choose **Test**. When the test finishes, you receive the following message:



Example Code

Use the following code to create the example.

Python Lambda Function

The following is the contents of the Python Lambda function. The Lambda function call the `TranslateText` operation and passes the review, the source language, and the target language to get the translated review. Save this file as `example.py` and them compress it in a .zip archive called `example.zip`. Save the file in an S3 bucket in the same region that you are running the example..

```
import logging
import json
import boto3
import os

translate = boto3.client('translate')
dynamodb = boto3.client('dynamodb')
firehose = boto3.client('firehose')

TABLE_NAME = os.getenv('TABLE_NAME')

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    logger.info(event)

    if 'source_language' in event and 'target_language' in event and 'review' in event and
 'review_id' in event:
        review_id = event['review_id']
        source_language = event['source_language']
        target_language = event['target_language']
        review = event['review']
```

```
        try:
            # The Lambda function queries the Amazon DynamoDB table to check whether
            # the review has already been translated. If the translated review
            # is already stored in Amazon DynamoDB, the function returns it.
            response = dynamodb.get_item(
                TableName=TABLE_NAME,
                Key={
                    'review_id': {
                        'N': review_id,
                    },
                    'language': {
                        'S': target_language,
                    },
                }
            )
            logger.info(response)
            if 'Item' in response:
                return response['Item']['review']['S']
        except Exception as e:
            logger.error(response)
            raise Exception("[ErrorMessage]: " + str(e))

        try:
            # The Lambda function calls the TranslateText operation and passes the
            # review, the source language, and the target language to get the
            # translated review.
            result = translate.translate_text(Text=review,
SourceLanguageCode=source_language, TargetLanguageCode=target_language)
            logging.info("Translation output: " + str(result))
        except Exception as e:
            logger.error(response)
            raise Exception("[ErrorMessage]: " + str(e))

        try:
            # After the review is translated, the function stores it using
            # the Amazon DynamoDB putItem operation. Subsequent requests
            # for this translated review are returned from Amazon DynamoDB.
            response = dynamodb.put_item(
            TableName=TABLE_NAME,
            Item={
                'review_id': {
                    'N': review_id,
                },
                'language': {
                    'S': target_language,
                },
                'review': {
                    'S': result.get('TranslatedText')
                }
                }
            )
            logger.info(response)
        except Exception as e:
                logger.error(e)
                raise Exception("[ErrorMessage]: " + str(e))
        return result.get('TranslatedText')
    else:
        logger.error(e)
        raise Exception("[ErrorMessage]: Invalid input ")
```

AWS CloudFormation Template

The following is the template file that you use with AWS CloudFormation to create and configure the Lambda function and the DynamoDB tables. Use this file when you create the AWS CloudFormation

stack for the example. Update `BUCKET_NAME` to the name of the S3 bucket that contains the `example.zip` file and then save it to a local directory as `template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:  ReviewTranslate:
    Type: 'AWS::Serverless::Function'
    Properties:
        Handler: example.lambda_handler
        Runtime: python2.7
        CodeUri:
          Bucket: BUCKET_NAME
          Key: example.zip
        Policies:
         - AWSLambdaFullAccess
         - TranslateReadOnly
        Environment:
          Variables:
            TABLE_NAME: !Ref ReviewTable
        Tracing: "Active"
  ReviewTable:
    Type: 'AWS::DynamoDB::Table'
    Properties:
        AttributeDefinitions:
            - AttributeName: "review_id"
              AttributeType: "N"
            - AttributeName: "language"
              AttributeType: "S"
        KeySchema:
            - AttributeName: "review_id"
              KeyType: "HASH"
            - AttributeName: "language"
              KeyType: "RANGE"
        ProvisionedThroughput:
            ReadCapacityUnits: 5
            WriteCapacityUnits: 5
```

Using Amazon Translate to Translate a Web Page

You can use Amazon Translate to translate the contents of a Web page. The following Java program translates a specified Web page from English to Spanish and creates an HTML file that contains the result of the translation. There are two functions in the program:

- A function that reads data from the source Web page, separates it into HTML elements, and then calls the second function to translate the element. At the end of the document, it writes the results to an HTML file.
- A function that calls the Amazon Translate service to translate the contents of an HTML element.

This example works on simple HTML pages without nested elements.

To configure the example

1. Install and configure the AWS SDK for Java. For instructions for installing the SDK for Java, see Set up the AWS SDK for Java.
2. Install the jsoup Java HTML parser. For instructions, see jsoup.
3. Create an IAM user with the minimum required permissions to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity*

and Access Management User Guide. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51).

4. Set up the credentials needed to run the sample. For instructions, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java Developer Guide*.
5. Create a new project in your Java IDE and copy the source code.
6. Change the region and endpoint to the region where you want to run the Amazon Translate operation. For a list of supported regions for Amazon Translate, see AWS Regions and Endpoints in the *AWS General Reference*.

```
package com.amazonaws.translateweb;

import com.amazonaws.auth.AWSCredentialsProviderChain;
import com.amazonaws.auth.EnvironmentVariableCredentialsProvider;
import com.amazonaws.auth.SystemPropertiesCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.translate.AmazonTranslate;
import com.amazonaws.services.translate.AmazonTranslateClient;
import com.amazonaws.services.translate.model.TranslateTextRequest;
import com.amazonaws.services.translate.model.TranslateTextResult;
import com.amazonaws.AmazonServiceException;

import java.io.IOException;
import java.io.PrintWriter;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;

import org.jsoup.select.Elements;


public class TranslateWebPage {

   public static void main(String[] args) throws InterruptedException {

   // Define the URL of the HTML content to translate
   String url = "http://example.com/source.html";

   // Create credentials using a provider chain that will evaluate in order;
   // a) Any Java system properties
   // b) Any environment variables
   // c) Any profile file
   AWSCredentialsProviderChain DefaultAWSCredentialsProviderChain = new
 AWSCredentialsProviderChain(
         new SystemPropertiesCredentialsProvider(),
         new EnvironmentVariableCredentialsProvider(),
         new ProfileCredentialsProvider()
      );

   // Create an endpoint configuration for the Translate service
   AwsClientBuilder.EndpointConfiguration endpointConfiguration = new
 AwsClientBuilder.EndpointConfiguration(
      "endpoint", "region");

   // Create a client for the Translate service
   AmazonTranslate translate = AmazonTranslateClient.builder()
      .withCredentials(DefaultAWSCredentialsProviderChain)
      .withEndpointConfiguration(endpointConfiguration).build();


   // Record the beginning of translating the HTML content at the url
```

```
   System.out.println("Translating URL: " + url);

   // Create an empty HTML document to store the parsed data
   Document doc;

   try {
      // Retrieve the HTML located at the URL
      doc = Jsoup.connect(url).get();

      // Select all of the elements in the HTML
      Elements eles = doc.select("*");

      // For each element
      for (Element ele : eles) {

          // Translate the element
          translateElement(ele, translate);

          // If you encounter service throttling when translating large web
          // pages, you can request a service limit increase. For details,
          // see https://aws.amazon.com/premiumsupport/knowledge-center/manage-service-
limits/,
          // or you can throttle your requests by inserting a sleep statement.
          // Thread.sleep(1000);
      }

      // Configure an output file for the translated HTML
      String fname = "output HTML file name";
      PrintWriter pw = new PrintWriter(fname, "UTF-8");

      // Write our translated HTML to the output file
      pw.println(doc);
      pw.close();

      // Record that the file has been saved
      System.out.println("Saved file "+fname);


      // Catch any exceptions in retrieving the HTML
   } catch (IOException e1) {
      e1.printStackTrace();
   }
}

   // This function is used to translate each individual element
   public static void translateElement(Element ele, AmazonTranslate translate) {

      // Check if the element has any text
      if (!ele.ownText().isEmpty()) {

         // Retrieve the text of the HTML element
         String text = ele.ownText();

         // Now translate the element's text
         try {

            // Translate from English to Spanish
            TranslateTextRequest request = new TranslateTextRequest()
               .withText(text)
               .withSourceLanguageCode("en")
               .withTargetLanguageCode("es");

            // Retrieve the result
            TranslateTextResult result  = translate.translateText(request);

            // Record the original and translated text
```

```
            System.out.println("Original text: " + text + " - Translated text: "+
 result.getTranslatedText());

            // Update the HTML element with the translated text
            ele.text(result.getTranslatedText());

            // Catch any translation errors
         } catch (AmazonServiceException e) {
                  System.err.println(e.getErrorMessage());
                  System.exit(1);
         }
      } else {
       // We have found a non-text HTML element. No action required.
      }
   }
}
```

Using Amazon Translate to Translate Large Documents

You can split large documents into smaller parts to keep the total document size below the document size limit. For more information about document size limits, see Service Limits (p. 66). The following Java program breaks long text documents into individual sentences and then translates each sentence from the source language to the target language. The program contains two sections:

- The `SentenceSegmenter` class that is responsible for breaking the source string into individual sentences. The sample uses the Java `BreakIterator` class.

- The `main` function that calls the `Translate` operation for each sentence in the source string. The `main` function also handles authentication with Amazon Translate.

To configure the example

1. Install and configure the AWS SDK for Java. For instructions for installing the SDK for Java, see Set up the AWS SDK for Java.
2. Create an IAM user with the minimum required permissions to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity and Access Management User Guide*. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51).
3. Set up the credentials needed to run the sample. For instructions, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java Developer Guide*.
4. Create a new project in your Java IDE and copy the source code.
5. Change the region to the region where you want to run the Amazon Translate operation. For a list of supported regions for Amazon Translate, see AWS Regions and Endpoints in the *AWS General Reference*.
6. Change the source and target languages to the languages to translate between.
7. Run the sample to see the translated text on standard output.

```
import com.amazonaws.auth.AWSCredentialsProviderChain;
import com.amazonaws.auth.EnvironmentVariableCredentialsProvider;
import com.amazonaws.auth.SystemPropertiesCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
```

```
import com.amazonaws.services.translate.AmazonTranslate;
import com.amazonaws.services.translate.AmazonTranslateClient;
import com.amazonaws.services.translate.model.TranslateTextRequest;
import com.amazonaws.services.translate.model.TranslateTextResult;
import java.text.BreakIterator;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class MultiSentenceTranslator {

    public static void main(String[] args) {
        // Define the text to be translated here
        String region = "region";
        String text = "Text to be translated";

        String sourceLang = "source language";
        String targetLang = "target language";

        // Break text into sentences
        SentenceSegmenter sentenceSegmenter = new SentenceSegmenter();
        List<String> sentences = new ArrayList<>();
        try {
            sentences = sentenceSegmenter.segment(text, sourceLang);
        } catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }

        // Create credentials using a provider chain that will evaluate in order;
        // a) Any Java system properties
        // b) Any environment variables
        // c) Any profile file
        AWSCredentialsProviderChain DefaultAWSCredentialsProviderChain = new
 AWSCredentialsProviderChain(
                new SystemPropertiesCredentialsProvider(),
                new EnvironmentVariableCredentialsProvider(),
                new ProfileCredentialsProvider()
        );

        // Create an Amazon Translate client
        AmazonTranslate translate = AmazonTranslateClient.builder()
                .withCredentials(DefaultAWSCredentialsProviderChain)
                .withRegion(region)
                .build();

        // Translate sentences and print the results to stdout
        for (String sentence : sentences) {
            TranslateTextRequest request = new TranslateTextRequest()
                    .withText(sentence)
                    .withSourceLanguageCode(sourceLang)
                    .withTargetLanguageCode(targetLang);
            TranslateTextResult result = translate.translateText(request);
            System.out.println("Original text: " + sentence);
            System.out.println("Translated text: " + result.getTranslatedText());
        }
    }

}

class SentenceSegmenter {
    public List<String> segment(final String text, final String lang) throws Exception {
        List<String> res = new ArrayList<>();
        BreakIterator sentenceIterator = BreakIterator.getSentenceInstance(new
 Locale(lang));
        sentenceIterator.setText(text);
```

```
        int prevBoundary = sentenceIterator.first();
        int curBoundary = sentenceIterator.next();
        while (curBoundary != BreakIterator.DONE) {
            String sentence = text.substring(prevBoundary, curBoundary);
            res.add(sentence);
            prevBoundary = curBoundary;
            curBoundary = sentenceIterator.next();
        }
        return res;
    }

}
```

Using Signature Version 4 with Amazon Translate

This example Python program shows how to use Signature Version 4 to add authentication information to Amazon Translate requests. The example makes a POST request, creates a JSON structure that contains the text to be translated in the body (payload) of the request, and passes authentication information in an Authorization header. For more information about using Signature Version 4, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

Setting Up

To run the example, perform the following steps:

1. Install the AWS Command Line Interface (AWS CLI). The AWS SDK for Python (Boto) is included when you install the AWS CLI. For instructions, see Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 7).
2. Create an AWS Identity and Access Management (IAM) user with the minimum required permission policy to run this example. For information about creating an IAM user, see Creating an IAM User in Your AWS Account in the *AWS Identity and Access Management User Guide*. For the required permissions policies, see Amazon Translate Identity-Based Policies (p. 51). Record the user access key ID and the secret access key.
3. Place the access key ID and secret access key in environment variables named `AWS_ACCESS_KEY and AWS_SECRET_ACCESS_KEY`, respectively. As a best practice, we recommend that you don't embed credentials in code.
4. Create a new file on your computer, copy the code for the example (which you can find in the next section), paste it into the file, and save the file with the extension `.py`.
5. In the code, replace `region` with the name of the AWS Region where you want to run the Amazon Translate `TranslateText` operation. For a list of supported Regions, see AWS Regions and Endpoints in the *AWS General Reference*.

Code

The following is the complete code of the example Python program.

After creating request values such as the endpoint URL and the body of the request, the code does the following:

1. Create a canonical request to the Amazon Translate `TranslateText` operation.
2. Create the string to that you hash to create the signature.
3. Calculate the signature.

4. Add the signature to the request header.
5. Send the request to the `TranslateText` operation.

To run the example on your computer, copy the code to a Python file.

```
# AWS Version 4 signing example

# Translate API (TranslateText)

# For more information about using Signature Version 4, see http://docs.aws.amazon.com/
general/latest/gr/sigv4_signing.html.
# This example makes a POST request to Amazon Translate and
# passes the text to translate JSON in the body (payload)
# of the request. Authentication information is passed in an
# Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests

# ************* REQUEST VALUES *************
method = 'POST'
service = 'translate'
region = 'region'
host = service + '.' + region + '.amazonaws.com'
endpoint = 'https://' + host + '/'

# POST requests use a content type header. For Amazon Translate,
# the content is JSON.
content_type = 'application/x-amz-json-1.1'
# Amazon Translate requires an x-amz-target header that has this format:
#     AWSShineFrontendService_20170701.<operationName>.
amz_target = 'AWSShineFrontendService_20170701.TranslateText'

# Pass request parameters for the TranslateText operation in a JSON block.
request_parameters =  '{'
request_parameters +=  '"Text": "Hello world.",'
request_parameters +=  '"SourceLanguageCode": "en",'
request_parameters +=  '"TargetLanguageCode": "de"'
request_parameters +=  '}'

# The following functions derive keys for the request. For more information, see
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python.
def sign(key, msg):
    return hmac.new(key, msg.encode("utf-8"), hashlib.sha256).digest()

def getSignatureKey(key, date_stamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), date_stamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Python can read the AWS access key from environment variables or the configuration file.
# In this example, keys are stored in environment variables. As a best practice, do not
# embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print 'No access key is available.'
    sys.exit()

# Create a timestamp for headers and the credential string.
t = datetime.datetime.utcnow()
amz_date = t.strftime('%Y%m%dT%H%M%SZ')
```

```
date_stamp = t.strftime('%Y%m%d') # The date without time is used in the credential scope.


# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# For information about creating a canonical request, see http://docs.aws.amazon.com/
general/latest/gr/sigv4-create-canonical-request.html.

# Step 1: Define the verb (GET, POST, etc.), which you have already done.

# Step 2: Create a canonical URI. A canonical URI is the part of the URI from domain to
 query.
# string (use '/' if no path)
canonical_uri = '/'

## Step 3: Create the canonical query string. In this example, request
# parameters are passed in the body of the request and the query string
# is blank.
canonical_querystring = ''

# Step 4: Create the canonical headers. Header names must be trimmed,
# lowercase, and sorted in code point order from low to high.
# Note the trailing \n.
canonical_headers = 'content-type:' + content_type + '\n' + 'host:' + host + '\n' + 'x-amz-
date:' + amz_date + '\n' + 'x-amz-target:' + amz_target + '\n'

# Step 5: Create the list of signed headers by listing the headers
# in the canonical_headers list, delimited with ";" and in alphabetical order.
# Note: The request can include any headers. Canonical_headers and
# signed_headers should contain headers to include in the hash of the
# request. "Host" and "x-amz-date" headers are always required.
# For Amazon Translate, content-type and x-amz-target are also required.
signed_headers = 'content-type;host;x-amz-date;x-amz-target'

# Step 6: Create the payload hash. In this example, the request_parameters
# variable contains the JSON request parameters.
payload_hash = hashlib.sha256(request_parameters).hexdigest()

# Step 7: Combine the elements to create a canonical request.
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n' +
 canonical_headers + '\n' + signed_headers + '\n' + payload_hash


# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Set the algorithm variable to match the hashing algorithm that you use, either SHA-256
 (recommended) or SHA-1.
#
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = date_stamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amz_date + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request).hexdigest()


# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the getSignaturKey function defined above.
signing_key = getSignatureKey(secret_key, date_stamp, region, service)

# Sign the string_to_sign using the signing_key.
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()


# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# Put the signature information in a header named Authorization.
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
```

```
# For Amazon Translate, the request can include any headers, but it must include "host,"
 "x-amz-date,"
# "x-amz-target," "content-type," and "Authorization" headers. Except for the authorization
# header, the headers must be included in the canonical_headers and signed_headers values,
 as
# noted earlier. Header order is not significant.
# Note: The Python 'requests' library automatically adds the 'host' header.
headers = {'Content-Type':content_type,
           'X-Amz-Date':amz_date,
           'X-Amz-Target':amz_target,
           'Authorization':authorization_header}


# ************* TASK 5: SEND THE REQUEST *************
print 'Request:\n\t' + request_parameters

response = requests.post(endpoint, data=request_parameters, headers=headers)
print 'Response:\n\t' + response.text
```

Security in Amazon Translate

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon Translate, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors, including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This topic helps you understand how to apply the shared responsibility model when using AWS. The following topics show you how to configure AWS to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS resources.

Topics

Data Protection in Amazon Translate

Amazon Translate conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all of the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon Simple Storage Service (Amazon S3).

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon Translate or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon Translate or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Topics

Encryption at Rest

Amazon Translate uses the default Amazon S3 key (SSE-S3) for server-side encryption of translations placed in your S3 bucket.

Amazon Translate uses an Amazon Elastic Block Store (Amazon EBS) volume encrypted with the default key.

Key Management

The default Amazon S3 key is managed by AWS. It is the responsibility of the customer to manage any customer-provided AWS Key Management Service (AWS KMS) keys.

Identity and Access Management for Amazon Translate

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Translate resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Troubleshooting Amazon Translate Identity and Access (p. 56)
- Amazon Translate API Permissions: Actions, Resources, and Conditions Reference (p. 58)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon Translate.

Service user – If you use the Amazon Translate service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Translate features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Translate, see Troubleshooting Amazon Translate Identity and Access (p. 56).

Service administrator – If you're in charge of Amazon Translate resources at your company, you probably have full access to Amazon Translate. It's your job to determine which Amazon Translate features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Translate, see How Amazon Translate Works with IAM (p. 51).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Translate. To view example Amazon Translate identity-based policies that you can use in IAM, see Amazon Translate Identity-based Policy Examples (p. 53).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then

securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests.

This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists (ACLs)

Access control policies (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they are the only policy type that does not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon

VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon Translate Works with IAM

You can manage access to Amazon Translate operations and resources with AWS Identity and Access Management. Before you use AWS Identity and Access Management (IAM) to manage access to Amazon Translate, you should understand which IAM features are available to use with Amazon Translate. To get a high-level view of how Amazon Translate and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

Amazon Translate Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon Translate supports specific actions,

resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon Translate use the following prefix before the action: `translate:`. For example, to grant someone permission to perform a translation with the Amazon Translate `TranslateText` API operation, log in as an admin and include the `translate:TranslateText` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon Translate defines its own set of actions that describe tasks that you can perform with this service. To see a list of Amazon Translate actions, see Actions Defined by Amazon Translate in the *IAM User Guide*.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "translate:action1",
      "translate:action2"
```

Resources

Amazon Translate doesn't support specifying resource Amazon Resource Names (ARNs) in a policy.

Condition Keys

Amazon Translate doesn't provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the *IAM User Guide*.

Examples

To see examples of Amazon Translate identity-based policies, see Amazon Translate Identity-based Policy Examples (p. 53).

Amazon Translate Resource-Based Policies

Amazon Translate doesn't support resource-based policies.

Authorization Based on Amazon Translate Tags

Amazon Translate doesn't support tagging resources or controlling access based on tags.

Amazon Translate IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon Translate

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS Security Token Service (AWS STS) API operations such as AssumeRole or GetFederationToken.

Amazon Translate supports using temporary credentials.

Service-Linked Roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit. the permissions for service-linked roles.

Amazon Translate doesn't support service-linked roles.

Amazon Translate Identity-based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon Translate resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specific resources that they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using the following example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Identity-based Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon Translate resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon Translate quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using the Amazon Translate Console

To access the Amazon Translate console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Translate resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon Translate console, also attach the following AWS managed policy to the entities.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "translate:*",
                "comprehend:DetectDominantLanguage",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics"
             ],
            "Resource": "*"

        }
    ]
}
```

The policy has four statements. The first grants permissions to use the `TranslateText` operation. The second grants permissions to the Amazon Comprehend `DetectDominantLanguage` operation to enable automatic language detection. The final two permissions grant permissions to Amazon CloudWatch to provide metrics support.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform. For more information, see Adding Permissions to a User in the *IAM User Guide*.

Permissions for Using Customer Managed Keys with Custom Terminologies

If you use AWS Key Management Service (AWS KMS) customer-managed keys (CMKs) with Amazon Translate custom terminologies, you might need additional permissions in your AWS KMS key policy.

To call the `ImportTerminology` operation with an AWS KMS CMK, add the following permissions to your existing AWS KMS key policy.

```
{
    "Id": "key-consolepolicy-3",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow access for use with Amazon Translate",
```

```
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM USER OR ROLE ARN"
            },
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:RetireGrant"
            ],
            "Resource": "*"
        }
    ]
}
```

To call the `GetTerminology` operation for a custom terminology that was imported with an KMS CMK, add the following permissions in the AWS KMS key policy.

```
{
    "Id": "key-consolepolicy-3",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow access for use with Amazon Translate",
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM USER OR ROLE ARN"
            },
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

To call the `ListTerminologies` or `DeleteTermionlogy` operations for a custom terminology that was imported with an AWS KMS CMK, you don't need to have any special AWS KMS permissions.

To use CMKs with all custom terminologies operations, add the following permissions in the AWS KMS key policy.

```
{
    "Id": "key-consolepolicy-3",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow access for use with Amazon Translate",
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM USER OR ROLE ARN"
            },
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:RetireGrant",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

For the Amazon Translate API actions and the resources that they apply to, see Amazon Translate API Permissions: Actions, Resources, and Conditions Reference (p. 58).

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ViewOwnUserInfo",
               "Effect": "Allow",
               "Action": [
                   "iam:GetUserPolicy",
                   "iam:ListGroupsForUser",
                   "iam:ListAttachedUserPolicies",
                   "iam:ListUserPolicies",
                   "iam:GetUser"
               ],
               "Resource": [
                   "arn:aws:iam::*:user/${aws:username}"
               ]
           },
           {
               "Sid": "NavigateInConsole",
               "Effect": "Allow",
               "Action": [
                   "iam:GetGroupPolicy",
                   "iam:GetPolicyVersion",
                   "iam:GetPolicy",
                   "iam:ListAttachedGroupPolicies",
                   "iam:ListGroupPolicies",
                   "iam:ListPolicyVersions",
                   "iam:ListPolicies",
                   "iam:ListUsers"
               ],
               "Resource": "*"
           }
       ]
   }
```

Troubleshooting Amazon Translate Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Translate and AWS Identity and Access Management (IAM).

Topics

- I Want to Allow People Outside of My AWS Account to Access My Amazon Translate Resources (p. 58)

I Am Not Authorized to Perform an Action in Amazon Translate

If the AWS Management Console tells you that you're not authorized to perform an action, contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

For example, the following error occurs when the `mateojackson` IAM user tries to use the console to translate text but doesn't have the `translate:TranslateText` permission.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 translate:TranslateText
```

Mateo would ask his administrator to update his policies to allow him to use the `translate:TranslateText` action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon Translate.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Translate. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an Administrator and Want to Allow Others to Access Amazon Translate

To allow others to access Amazon Translate, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon Translate.

To get started right away, see Creating Your First IAM Delegated User and Group in the *IAM User Guide*.

I Want to Allow People Outside of My AWS Account to Access My Amazon Translate Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Translate supports these features, see How Amazon Translate Works with IAM (p. 51).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Amazon Translate API Permissions: Actions, Resources, and Conditions Reference

Use the following table as a reference when writing a permissions policy that you can attach to an IAM identity (an identity-based policy). The list includes each Amazon Translate API operation, the corresponding action for which you can grant permissions, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

To express conditions, you can use AWS-wide condition keys in your Amazon Translate policies. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

> **Note**
> To specify an action, use the `translate:` prefix followed by the API operation name, for example, `translate:TranslateText`.

Monitoring Amazon Translate

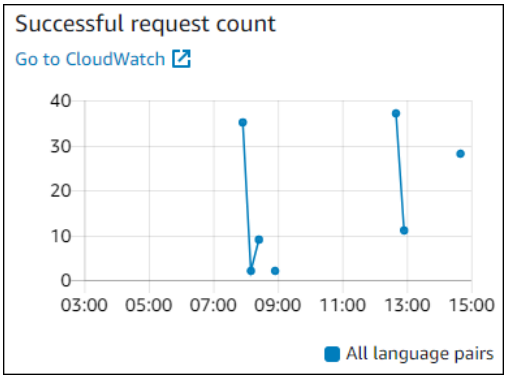
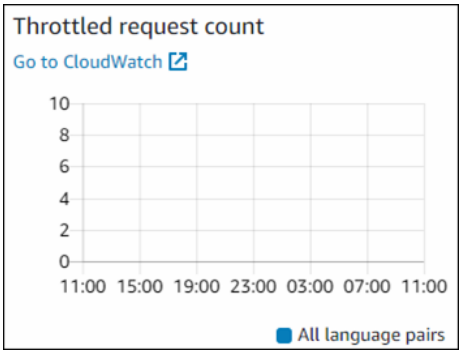
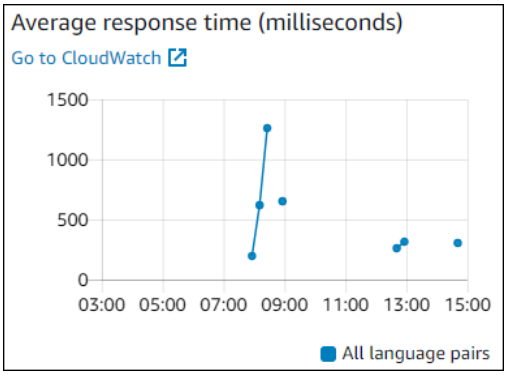
Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Translate and your solutions. AWS provides various tools that you can use to monitor Amazon Translate.

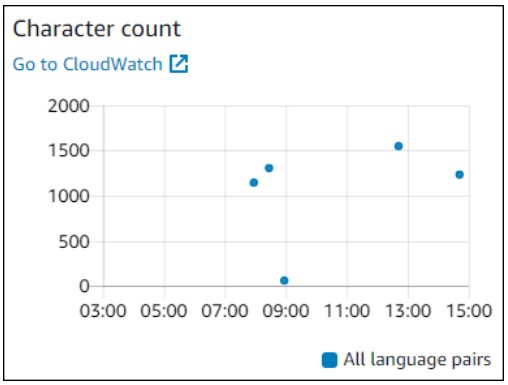
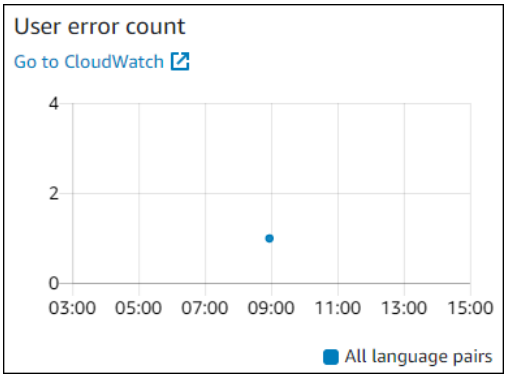
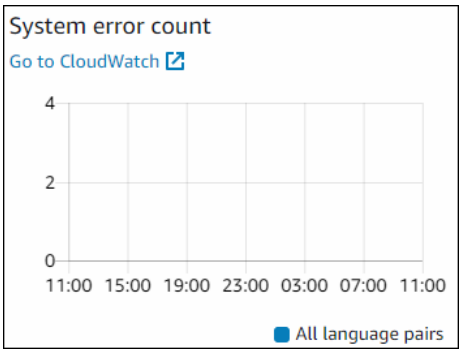
You can configure some of these tools to monitor your solutions for you. We recommend that you automate monitoring tasks as much as possible.

Amazon Translate provides preconfigured graphs that show you the most important metrics for your solution. Each graph offers a window into your solution's performance. To get different views of how your solution is performing over time, you can change the time range that the graphs show.

You can also use Amazon CloudWatch to monitor Amazon Translate. With CloudWatch, you can automate monitoring specific metrics for your solutions. You receive a notice whenever a metric is outside of the thresholds that you set. You can also use the CloudWatch API to create a custom monitoring application that is suitable for your needs. For more information, see What is Amazon CloudWatch in the *Amazon CloudWatch User Guide*.

The following table describes each of the preconfigured graphs provided by Amazon Translate.

Graph	Description
	Successful request count The number of successful requests made to Amazon Translate during the specified time period.
	Throttled request count The number of requests to Amazon Translate that were throttled during the specified time period. Use this information to determine if your application is sending requests to Amazon Translate too quickly.
	Average response time The average length of time that it took Amazon Translate to process your request during the specified time period.

Graph	Description
	Character count The total number of characters that you sent to Amazon Translate during the specified time period. This is the number of characters that you will be billed for.
	User error count The number of user errors that occurred during the specified time period. User errors are in the HTTP error code range 400-499.
	System error count The number of system errors that occurred during the specified time period. System errors are in the HTTP error code range 500-599.

Monitoring Amazon Translate

With Amazon CloudWatch, you can get metrics for individual Amazon Translate operations or global Amazon Translate metrics for your account. Use metrics to track the health of your Amazon Translate solutions and to set up alarms to notify you when one or more metrics fall outside a defined threshold. For example, you can monitor the number of requests made to Amazon Translate in a particular time period, see the latency of requests, or raise an alarm when errors exceed a threshold.

Understanding CloudWatch Metrics for Amazon Translate

To get metrics for your Amazon Translate operations, you specify the following information:

- The metric dimension. A *dimension* is a set of name-value pairs that you use to identify a metric. Amazon Translate has two dimensions:
  - `Operation`
  - `Language pair`
- The metric name, such as `SuccessfulRequestCount` or `RequestCharacters`. For a complete list of metrics, see CloudWatch Metrics for Amazon Translate (p. 63).

You can get metrics for Amazon Translate with the AWS Management Console, the AWS CLI, or the CloudWatch API. You can use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools.

The following table lists some common uses for CloudWatch metrics. These are suggestions to get you started, not a comprehensive list.

How Do I?	Monitor This Metric
Track the number of successful requests	The `sum` statistic of the `SuccessfulRequestCount` metric
Know if my application has reached its maximum throughput	The `sum` statistic of the `ThrottledCount` metric
Find the response time for my application	The `average` statistic of the `ResponseTime` metric
Find the number of errors for my application	The `sum` statistic of the `ServerErrorCount` and `UserErrorCount` metrics
Find the number of billable characters	The `sum` statistic of the `CharacterCount` metric

You must have the appropriate CloudWatch permissions to monitor Amazon Translate with CloudWatch For more information, see Authentication and Access Control for Amazon CloudWatch in the *Amazon CloudWatch User Guide*.

Viewing Amazon Translate Metrics

View Amazon Translate metrics in the CloudWatch console.

To view metrics (CloudWatch console)

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, choose **All Metrics**, and then choose **AWS/Translate**.
3. Choose the dimension, choose a metric name, and choose **Add to graph**.
4. Choose a value for the date range. The metric count for the specified date range is displayed in the graph.

Logging Amazon Translate API Calls with AWS CloudTrail

Amazon Translate is integrated with AWS CloudTrail, a service that provides a record of actions taken by an IAM user, IAM role, or AWS service in Amazon Translate. CloudTrail captures all API calls for Amazon

Translate as events. This includes calls from the Amazon Translate console and code calls to the Amazon Translate API operations. If you create a CloudTrail trail, you can enable continuous delivery of CloudTrail events, including events for Amazon Translate, to an Amazon Simple Storage Service (Amazon S3) bucket. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. You can use the information collected by CloudTrail to determine the request that was made to Amazon Translate, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Topics

Amazon Translate Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Translate, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Translate, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail with the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. You can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon Translate actions are logged by CloudTrail and are documented in the API reference section. For example, calls to the `DeleteTerminology`, `ImportTerminology` and `TranslateText` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. This information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity Element.

Understanding Amazon Translate Log File Entries

A *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request

parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `TranslateText` action.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/Administrator",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2019-09-03T20:32:50Z",
    "eventSource": "translate.amazonaws.com",
    "eventName": "TranslateText",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.207 Python/3.4.7
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.12.197",
    "requestParameters": {
        "text": "Amazon is based in Seattle.",
        "sourceLanguageCode": "en",
        "targetLanguageCode": "fr"
    },
    "responseElements": {
        "translatedText": "Amazon est basé à Seattle.",
        "sourceLanguageCode": "en",
        "targetLanguageCode": "fr"
    },
    "requestID": "f56da956-284e-4983-b6fc-59befa20e2bf",
    "eventID": "1dc75278-84d7-4bb2-861a-493d08d67391",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


CloudWatch Metrics and Dimensions for Amazon Translate

To monitor your solution's performance, use the Amazon CloudWatch metrics and dimensions for Amazon Translate.

CloudWatch Metrics for Amazon Translate

Metric	Description
`CharacterCount`	The number of billable characters in requests. Valid dimensions: Language pair, Operation Valid statistics: Average, Maximum, Minimum, Sum Unit: Count
`ResponseTime`	The time that it took to respond to a request. Valid dimensions: Language pair, Operation

Metric	Description
	Valid statistics: Data samples, Average Unit: For Data samples, count. For Average statistics, milliseconds.
`ServerErrorCount`	The number of server errors. The HTTP response code range for a server error is 500 to 599. Valid dimension: Operation Valid statistics: Average, Sum Unit: Count
`SuccessfulRequestCount`	The number of successful translation requests. The response code for a successful request is 200 to 299. Valid dimension: Operation Valid statistics: Average, Sum Unit: Count
`ThrottledCount`	The number of requests subject to throttling. Use `ThrottledCount` to determine if your application is sending requests to Amazon Translate faster than your account is configured to accept them. For more information, see Amazon Translate Limits in the *Amazon Web Services General Reference*. Valid dimension: Operation Valid statistics: Average, Sum Unit: Count
`UserErrorCount`	The number of user errors that occurred. The HTTP response code range for a user error is 400 to 499. Valid dimension: Operation Valid statistics: Average, Sum Unit: Count

CloudWatch Dimensions for Amazon Translate

Use the following dimensions to filter Amazon Translate metrics. Metrics are grouped by the source language and the target language.

Dimension	Description
`LanguagePair`	Restricts the metrics to only those that contain the specified languages.
`Operation`	Restricts the metrics to only those with the specified operation.

Compliance Validation for Amazon Translate

Third-party auditors assess the security and compliance of Amazon Translate as part of multiple AWS compliance programs. These include PCI, FedRAMP, HIPAA, and others. You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon Translate is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

Resilience in Amazon Translate

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure Security in Amazon Translate

As a managed service, Amazon Translate is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

To access Amazon Translate through the network, you use AWS published API calls. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems, such as Java 7 and later, support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an AWS Identity and Access Management (IAM) principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Guidelines and Limits

The following sections contain information about Amazon Translate guidelines and limits.

Topics

Supported AWS Regions

For a list of AWS Regions that support Amazon Translate, see the AWS Region Table or AWS Regions and Endpoints in the *Amazon Web Services General Reference*.

Compliance

For more information about Amazon Translate compliance programs, see AWS Compliance, AWS Compliance Programs, and AWS Services in Scope by Compliance Program.

Throttling

For information about throttling for Amazon Translate or to request a limit increase, see Amazon Translate Limits in the *Amazon Web Services General Reference*.

Guidelines

To continuously improve the quality of its analysis models, Amazon Translate might store your data. To learn more, see the Amazon Translate FAQ.

You can request that we delete your data and that future data associated with your account isn't stored by contacting AWS Support. However, because deleting your data can also delete unique training data that is helpful in improving translation, doing so might reduce the quality of your translations.

Service Limits

Amazon Translate has the following service limitations.

Description	Limit
General Limits	

Description	Limit
Character encoding	UTF-8
Document size (UTF-8 characters)	5,000 bytes
Custom Terminology Limits	
Custom terminology file size	10 Mb
Maximum number of target languages per custom terminology file	10
Maximum number of custom terminologies per AWS account per AWS Region	100
Maximum source and target text length per custom terminology term	200 bytes
Maximum number of terms from a custom terminology source text applied to a `TranslateText` request	256*

*Amazon Translate uses the first 256 terms matched in a termonology source file. This limit applies across all terminologies used in a `TranslateText` request. At this time, you can use a maximum of 1 terminology file per `TranslateText` request.

Document History for Amazon Translate

The following table describes the documentation for this release of Amazon Translate.

- **Latest documentation update:** May 8th, 2019

update-history-change	update-history-description	update-history-date
New languages	Amazon Translate adds new languages for translation: Greek, Hungarian, Romanian, Thai, Ukrainian, Urdu, and Vietnamese. For a list of the language combinations that Amazon Translate can translate directly, see Supported Languages.	October 3, 2019
New feature	Amazon Translate adds FedRAMP compliance. For more information, see Compliance.	July 31, 2019
New feature	Amazon Translate adds SOC compliance. For more information, see Compliance.	May 30, 2019
New regions	Amazon Translate adds support for the Asia Pacific (Mumbai), Asia Pacific (Singapore), Asia Pacific (Tokyo), and Canada (Central) Regions. For a complete list of the AWS Regions supported by Amazon Translate, see the AWS Region Table or AWS Regions and Endpoints in the *Amazon Web Services General Reference*.	May 8, 2019
New languages	Amazon Translate adds new languages for translation: Hindi, Malay, Norwegian, and Persian. For a list of the language combinations that Amazon Translate can translate directly, see Supported Languages.	May 6, 2019
New region	Amazon Translate adds support for the EU (Frankfurt) and Asia Pacific (Seoul) Regions. For a complete list of the AWS Regions supported by Amazon	February 28, 2019

	Translate, see the AWS Region Table or AWS Regions and Endpoints in the *Amazon Web Services General Reference*.	
New feature	Amazon Translate adds PCI compliance. For more information, see Compliance.	December 12, 2018
New feature	Amazon Translate adds four new APIs and the custom terminology feature to give you more control over your translation. By using a custom terminology with your translation requests, you can make sure that your brand names, character names, model names, and other unique content is translated exactly the way you want it, every time, regardless of the standard translation or context. For more information, see Custom Terminology.	November 27, 2018
New languages	Amazon Translate now translates documents in the following languages: Danish, Dutch, Finnish, Hebrew, Indonesian, Korean, Polish, and Swedish. Amazon Translate continues to improve direct translation by significantly reducing the number of unsupported language pairs. For the language combinations that Amazon Translate can translate directly, see Supported Languages.	November 20, 2018
New feature	Amazon Translate adds direct translation between supported languages other than English. For the language combinations that Amazon Translate can translate directly, see Supported Languages.	October 29, 2018
New feature	Amazon Translate adds HIPAA compliance. For more information, see Compliance.	October 25, 2018

New feature	Amazon Translate adds multiple new languages for translation: Chinese (Tradition), Czech, Italian, Japanese, Russian, and Turkish. For a list of languages that Amazon Translate supports, see Supported Languages.	July 17, 2018
New feature	Amazon Translate adds support for automatic source language detection. For more information, see How Amazon Translate Works.	April 4, 2018
New guide (p. 68)	This is the first release of the *Amazon Translate Developer Guide*.	November 29, 2017

API Reference

This section contains the API Reference documentation.

HTTP Headers

In addition to the usual HTTP headers, Amazon Translate operations have the required headers:

Header	Value	Description
Content-Type:	application/x-amz-json-1.1	Specifies that the request content is JSON. Also specifies the JSON version.
X-Amz-Date:	<Date>	The date used to create the signature in the Authorization header. The format must be ISO 8601 basic in the YYYYMMDD'T'HHMMSS'Z' format. For example, the following date/time 20180820T184626Z is a valid x-amz-date for use with Amazon Translate. For more information about using the Authorization header, see Using Signature Version 4 with Amazon Translate .
X-Amz-Target:	AWSShineFrontendService_20170701.<target>	The target Amazon Translate operation. For example, use `AWSShineFrontendService_20170701.Tra` to call the `TranslateText` operation.

Actions

The following actions are supported:

- DeleteTerminology (p. 72)
- GetTerminology (p. 74)
- ImportTerminology (p. 77)
- ListTerminologies (p. 80)
- TranslateText (p. 83)

DeleteTerminology

A synchronous action that deletes a custom terminology.

Request Syntax

```
{
   "Name": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 95).

The request accepts the following data in JSON format.

Name (p. 72)

The name of the custom terminology being deleted.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `^([A-Za-z0-9-]_?)+$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 93).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

ResourceNotFoundException

The resource you are looking for has not been found. Review the resource you're looking for and see if a different resource will accomplish your needs before retrying the revised request. .

HTTP Status Code: 400

TooManyRequestsException

You have made too many requests within a short period of time. Wait for a short time and then try your request again.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

GetTerminology

Retrieves a custom terminology.

Request Syntax

```
{
   "Name": "string",
   "TerminologyDataFormat": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 95).

The request accepts the following data in JSON format.

Name (p. 74)

The name of the custom terminology being retrieved.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `^([A-Za-z0-9-]_?)+$`

Required: Yes

TerminologyDataFormat (p. 74)

The data format of the custom terminology being retrieved, either CSV or TMX.

Type: String

Valid Values: `CSV | TMX`

Required: Yes

Response Syntax

```
{
   "TerminologyDataLocation": {
      "Location": "string",
      "RepositoryType": "string"
   },
   "TerminologyProperties": {
      "Arn": "string",
      "CreatedAt": number,
      "Description": "string",
      "EncryptionKey": {
         "Id": "string",
         "Type": "string"
      },
      "LastUpdatedAt": number,
      "Name": "string",
      "SizeBytes": number,
      "SourceLanguageCode": "string",
```

```
      "TargetLanguageCodes": [ "string" ],
      "TermCount": number
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TerminologyDataLocation (p. 74)

The data location of the custom terminology being retrieved. The custom terminology file is returned in a presigned url that has a 30 minute expiration.

Type: TerminologyDataLocation (p. 91) object

TerminologyProperties (p. 74)

The properties of the custom terminology being retrieved.

Type: TerminologyProperties (p. 92) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 93).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidParameterValueException

The value of the parameter is invalid. Review the value of the parameter you are using to correct it, and then retry your operation.

HTTP Status Code: 400

ResourceNotFoundException

The resource you are looking for has not been found. Review the resource you're looking for and see if a different resource will accomplish your needs before retrying the revised request. .

HTTP Status Code: 400

TooManyRequestsException

You have made too many requests within a short period of time. Wait for a short time and then try your request again.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ImportTerminology

Creates or updates a custom terminology, depending on whether or not one already exists for the given terminology name. Importing a terminology with the same name as an existing one will merge the terminologies based on the chosen merge strategy. Currently, the only supported merge strategy is OVERWRITE, and so the imported terminology will overwrite an existing terminology of the same name.

If you import a terminology that overwrites an existing one, the new terminology take up to 10 minutes to fully propagate and be available for use in a translation due to cache policies with the DataPlane service that performs the translations.

Request Syntax

```
{
   "Description": "string",
   "EncryptionKey": {
      "Id": "string",
      "Type": "string"
   },
   "MergeStrategy": "string",
   "Name": "string",
   "TerminologyData": {
      "File": blob,
      "Format": "string"
   }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 95).

The request accepts the following data in JSON format.

Description (p. 77)

The description of the custom terminology being imported.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `[\P{M}\p{M}]{0,256}`

Required: No

EncryptionKey (p. 77)

The encryption key for the custom terminology being imported.

Type: EncryptionKey (p. 88) object

Required: No

MergeStrategy (p. 77)

The merge strategy of the custom terminology being imported. Currently, only the OVERWRITE merge strategy is supported. In this case, the imported terminology will overwrite an existing terminology of the same name.

Type: String

Valid Values: `OVERWRITE`

Required: Yes

Name (p. 77)

The name of the custom terminology being imported.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `^([A-Za-z0-9-]_?)+$`

Required: Yes

TerminologyData (p. 77)

The terminology data for the custom terminology being imported.

Type: TerminologyData (p. 90) object

Required: Yes

Response Syntax

```
{
   "TerminologyProperties": {
      "Arn": "string",
      "CreatedAt": number,
      "Description": "string",
      "EncryptionKey": {
         "Id": "string",
         "Type": "string"
      },
      "LastUpdatedAt": number,
      "Name": "string",
      "SizeBytes": number,
      "SourceLanguageCode": "string",
      "TargetLanguageCodes": [ "string" ],
      "TermCount": number
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TerminologyProperties (p. 78)

The properties of the custom terminology being imported.

Type: TerminologyProperties (p. 92) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 93).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidParameterValueException

The value of the parameter is invalid. Review the value of the parameter you are using to correct it, and then retry your operation.

HTTP Status Code: 400

LimitExceededException

The specified limit has been exceeded. Review your request and retry it with a quantity below the stated limit.

HTTP Status Code: 400

TooManyRequestsException

You have made too many requests within a short period of time. Wait for a short time and then try your request again.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

ListTerminologies

Provides a list of custom terminologies associated with your account.

Request Syntax

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 95).

The request accepts the following data in JSON format.

MaxResults (p. 80)

The maximum number of custom terminologies returned per list request.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 500.

Required: No

NextToken (p. 80)

If the result of the request to ListTerminologies was truncated, include the NextToken to fetch the next group of custom terminologies.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "TerminologyPropertiesList": [
      {
         "Arn": "string",
         "CreatedAt": number,
         "Description": "string",
         "EncryptionKey": {
            "Id": "string",
            "Type": "string"
         },
         "LastUpdatedAt": number,
         "Name": "string",
         "SizeBytes": number,
         "SourceLanguageCode": "string",
```

```
        "TargetLanguageCodes": [ "string" ],
        "TermCount": number
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 80)

If the response to the ListTerminologies was truncated, the NextToken fetches the next group of custom terminologies.

Type: String

Length Constraints: Maximum length of 8192.

Pattern: `\p{ASCII}{0,8192}`

TerminologyPropertiesList (p. 80)

The properties list of the custom terminologies returned on the list request.

Type: Array of TerminologyProperties (p. 92) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 93).

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidParameterValueException

The value of the parameter is invalid. Review the value of the parameter you are using to correct it, and then retry your operation.

HTTP Status Code: 400

TooManyRequestsException

You have made too many requests within a short period of time. Wait for a short time and then try your request again.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

TranslateText

Translates input text from the source language to the target language. It is not necessary to use English (en) as either the source or the target language, but not all language combinations are supported by Amazon Translate. For more information, see Supported Languages and Language Codes (p. 1).

Request Syntax

```
{
   "SourceLanguageCode": "string",
   "TargetLanguageCode": "string",
   "TerminologyNames": [ "string" ],
   "Text": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 95).

The request accepts the following data in JSON format.

SourceLanguageCode (p. 83)

The language code for the language of the source text. The language must be a language supported by Amazon Translate. For a list of language codes, see Supported Languages and Language Codes (p. 1).

To have Amazon Translate determine the source language of your text, you can specify `auto` in the `SourceLanguageCode` field. If you specify `auto`, Amazon Translate will call Amazon Comprehend to determine the source language.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 5.

Required: Yes

TargetLanguageCode (p. 83)

The language code requested for the language of the target text. The language must be a language supported by Amazon Translate.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 5.

Required: Yes

TerminologyNames (p. 83)

The name of the terminology list file to be used in the TranslateText request. You can use 1 terminology list at most in a `TranslateText` request. Terminology lists can contain a maximum of 256 terms.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `^([A-Za-z0-9-]_?)+$`

Required: No

Text (p. 83)

The text to translate. The text string can be a maximum of 5,000 bytes long. Depending on your character set, this may be fewer than 5,000 characters.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 5000.

Pattern: `[\P{M}\p{M}]{1,5000}`

Required: Yes

Response Syntax

```
{
   "AppliedTerminologies": [
      {
         "Name": "string",
         "Terms": [
            {
               "SourceText": "string",
               "TargetText": "string"
            }
         ]
      }
   ],
   "SourceLanguageCode": "string",
   "TargetLanguageCode": "string",
   "TranslatedText": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AppliedTerminologies (p. 84)

The names of the custom terminologies applied to the input text by Amazon Translate for the translated text response.

Type: Array of AppliedTerminology (p. 87) objects

SourceLanguageCode (p. 84)

The language code for the language of the source text.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 5.

TargetLanguageCode (p. 84)

The language code for the language of the target text.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 5.

TranslatedText (p. 84)

The translated text.

Type: String

Length Constraints: Maximum length of 10000.

Pattern: `[\P{M}\p{M}]{0,10000}`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 93).

DetectedLanguageLowConfidenceException

The confidence that Amazon Comprehend accurately detected the source language is low. If a low confidence level is acceptable for your application, you can use the language in the exception to call Amazon Translate again. For more information, see the DetectDominantLanguage operation in the *Amazon Comprehend Developer Guide*.

HTTP Status Code: 400

InternalServerException

An internal server error occurred. Retry your request.

HTTP Status Code: 500

InvalidRequestException

The request that you made is invalid. Check your request to determine why it's invalid and then retry the request.

HTTP Status Code: 400

ResourceNotFoundException

The resource you are looking for has not been found. Review the resource you're looking for and see if a different resource will accomplish your needs before retrying the revised request. .

HTTP Status Code: 400

ServiceUnavailableException

The Amazon Translate service is temporarily unavailable. Please wait a bit and then retry your request.

HTTP Status Code: 500

TextSizeLimitExceededException

The size of the text you submitted exceeds the size limit. Reduce the size of the text or use a smaller document and then retry your request.

HTTP Status Code: 400

TooManyRequestsException

You have made too many requests within a short period of time. Wait for a short time and then try your request again.

HTTP Status Code: 400

UnsupportedLanguagePairException

Amazon Translate does not support translation from the language of the source text into the requested target language. For more information, see Exception Handling (p. 4).

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Data Types

The following data types are supported:

- AppliedTerminology (p. 87)
- EncryptionKey (p. 88)
- Term (p. 89)
- TerminologyData (p. 90)
- TerminologyDataLocation (p. 91)
- TerminologyProperties (p. 92)

AppliedTerminology

The custom terminology applied to the input text by Amazon Translate for the translated text response. This is optional in the response and will only be present if you specified terminology input in the request. Currently, only one terminology can be applied per TranslateText request.

Contents

Name

The name of the custom terminology applied to the input text by Amazon Translate for the translated text response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `^([A-Za-z0-9-]_?)+$`

Required: No

Terms

The specific terms of the custom terminology applied to the input text by Amazon Translate for the translated text response. A maximum of 250 terms will be returned, and the specific terms applied will be the first 250 terms in the source text.

Type: Array of Term (p. 89) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

EncryptionKey

The encryption key used to encrypt the custom terminologies used by Amazon Translate.

Contents

Id

The Amazon Resource Name (ARN) of the encryption key being used to encrypt the custom terminology.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 400.

Pattern: `(arn:aws((-us-gov)|(-iso)|(-iso-b)|(-cn))?:kms:)?([a-z]{2}-[a-z]+(-[a-z]+)?-\d:)?(\d{12}:)?(((key/)?[a-zA-Z0-9-_]+)|(alias/[a-zA-Z0-9:/_-]+))`

Required: Yes

Type

The type of encryption key used by Amazon Translate to encrypt custom terminologies.

Type: String

Valid Values: `KMS`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Term

The term being translated by the custom terminology.

Contents

SourceText

The source text of the term being translated by the custom terminology.

Type: String

Length Constraints: Maximum length of 10000.

Pattern: `[\P{M}\p{M}]{0,10000}`

Required: No

TargetText

The target text of the term being translated by the custom terminology.

Type: String

Length Constraints: Maximum length of 10000.

Pattern: `[\P{M}\p{M}]{0,10000}`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

TerminologyData

The data associated with the custom terminology.

Contents

File

The file containing the custom terminology data. Your version of the AWS SDK performs a Base64-encoding on this field before sending a request to the AWS service. Users of the SDK should not perform Base64-encoding themselves.

Type: Base64-encoded binary data object

Length Constraints: Maximum length of 10485760.

Required: Yes

Format

The data format of the custom terminology. Either CSV or TMX.

Type: String

Valid Values: `CSV | TMX`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

TerminologyDataLocation

The location of the custom terminology data.

Contents

Location

The location of the custom terminology data.

Type: String

Length Constraints: Maximum length of 10000.

Pattern: `[\P{M}\p{M}]{0,10000}`

Required: Yes

RepositoryType

The repository type for the custom terminology data.

Type: String

Length Constraints: Maximum length of 10000.

Pattern: `[\P{M}\p{M}]{0,10000}`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

TerminologyProperties

The properties of the custom terminology.

Contents

Arn

The Amazon Resource Name (ARN) of the custom terminology.

Type: String

Pattern: `^arn:aws((-us-gov)|(-iso)|(-iso-b)|(-cn))?:translate:[a-zA-Z0-9-]+:[0-9]{12}:terminology/.+?/.+?$`

Required: No

CreatedAt

The time at which the custom terminology was created, based on the timestamp.

Type: Timestamp

Required: No

Description

The description of the custom terminology properties.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `[\P{M}\p{M}]{0,256}`

Required: No

EncryptionKey

The encryption key for the custom terminology.

Type: EncryptionKey (p. 88) object

Required: No

LastUpdatedAt

The time at which the custom terminology was last update, based on the timestamp.

Type: Timestamp

Required: No

Name

The name of the custom terminology.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: `^([A-Za-z0-9-]_?)+$`

Required: No

SizeBytes

The size of the file used when importing a custom terminology.

Type: Integer

Required: No

SourceLanguageCode

The language code for the source text of the translation request for which the custom terminology is being used.

Type: String

Length Constraints: Minimum length of 2. Maximum length of 5.

Required: No

TargetLanguageCodes

The language codes for the target languages available with the custom terminology file. All possible target languages are returned in array.

Type: Array of strings

Length Constraints: Minimum length of 2. Maximum length of 5.

Required: No

TermCount

The number of terms included in the custom terminology.

Type: Integer

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination

Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString

The query string contains a syntax error.

HTTP Status Code: 404

MissingAction

The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: `AWS4-HMAC-SHA256`

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key*/*YYYYMMDD*/*region*/*service*/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: `20120325T120000Z`.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the *Amazon Web Services General Reference*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.