# Amazon Comprehend

## Developer Guide

# Amazon Comprehend: Developer Guide

# Table of Contents

# What Is Amazon Comprehend?

Amazon Comprehend uses natural language processing (NLP) to extract insights about the content of documents. Amazon Comprehend processes any text file in UTF-8 format. It develops insights by recognizing the entities, key phrases, language, sentiments, and other common elements in a document. Use Amazon Comprehend to create new products based on understanding the structure of documents. For example, using Amazon Comprehend you can search social networking feeds for mentions of products or scan an entire document repository for key phrases.

You work with one or more documents at a time to evaluate their content and gain insights about them. Some of the insights that Amazon Comprehend develops about a document include:

- **Entities** – Amazon Comprehend returns a list of entities, such as people, places, and locations, identified in a document. For more information, see Detect Entities (p. 50).
- **Key phrases** – Amazon Comprehend extracts key phrases that appear in a document. For example, a document about a basketball game might return the names of the teams, the name of the venue, and the final score. For more information, see Locate Key Phrases (p. 51).
- **Language** – Amazon Comprehend identifies the dominant language in a document. Amazon Comprehend can identify 100 languages. For more information, see Detect the Dominant Language (p. 48).
- **Sentiment** – Amazon Comprehend determines the emotional sentiment of a document. Sentiment can be positive, neutral, negative, or mixed. For more information, see Determine the Sentiment (p. 52).
- **Syntax** – Amazon Comprehend parses each word in your document and determines the part of speech for the word. For example, in the sentence "It is raining today is Seattle," "it" is identified as a pronoun, "raining" is identified as a verb, and "Seattle" is identified as a proper noun. For more information, see Analyze Syntax (p. 53).

## Comprehend Custom

Customize Comprehend for your specific requirements without the skillset required to build machine learning-based NLP solutions. Using automatic machine learning, or AutoML, Comprehend Custom builds customized NLP models on your behalf, using data you already have. Training and calling custom comprehend models are both async (batch) operations.

Custom Classification: Create custom document classifiers to organize you documents into your own categories. For each classification label, provide a set of documents that best represent that label and submit the training data as a CSV file. You can have multiple document classifiers, each trained on a different set of input documents. Once a classifier is trained it can be used on any number of unlabeled document sets. Customers can use the console for a code-free experience or install the latest AWS SDK. For more information, see Custom Classification (p. 66).

Custom Entities: Create custom entity types that analyze text for your specific terms and noun-based phrases. Customers can train a custom entity type to extract terms like policy number, or phrases that imply a customer escalation. To train the feature, customers need to provide a list of the entities (terms and phrases) and a set of documents that contain them, stored in S3. Once the model is trained, customers can submit analysis jobs against their model to extract their custom entities. For more information, see Custom Entity Recognition (p. 72).

# Document Clustering (Topic Modeling)

You can also use Amazon Comprehend to examine a corpus of documents to organize them based on similar keywords within them. Document clustering (topic modeling) is useful to organize a large corpus of documents into topics or clusters that are similar based on the frequency of words within them.

Topic modeling is a asynchronous process, you submit a set of documents for processing and then later get the results when processing is complete. Amazon Comprehend does topic modeling on large document sets, for best results you should include at least 1,000 documents when you submit a topic modeling job. For more information, see Topic Modeling (p. 55).

# Examples

The following examples show how you might use the Amazon Comprehend operations in your applications.

**Example 1: Find documents about a subject**

Find the documents about a particular subject using Amazon Comprehend topic modeling. Scan a set of documents to determine the topics discussed, and to find the documents associated with each topic. You can specify the number of topics that Amazon Comprehend should return from the document set.

**Example 2: Find out how customers feel about your products**

If your company publishes a catalog, let Amazon Comprehend tell you what customers think of your products. Send each customer comment to the `DetectSentiment` operation and it will tell you whether customers feel positive, negative, neutral, or mixed about a product.

**Example 3: Discover what matters to your customers**

Use Amazon Comprehend topic modeling to discover the topics that your customers are talking about on your forums and message boards, then use entity detection to determine the people, places, and things that they associate with the topic. Finally, use sentiment analysis to determine how your customers feel about a topic.

# Benefits

Some of the benefits of using Amazon Comprehend include:

- **Integrate powerful natural language processing into your apps**—Amazon Comprehend removes the complexity of building text analysis capabilities into your applications by making powerful and accurate natural language processing available with a simple API. You don't need textual analysis expertise to take advantage of the insights that Amazon Comprehend produces.
- **Deep learning based natural language processing**—Amazon Comprehend uses deep learning technology to accurately analyze text. Our models are constantly trained with new data across multiple domains to improve accuracy.
- **Scalable natural language processing**—Amazon Comprehend enables you to analyze millions of documents so that you can discover the insights that they contain.
- **Integrate with other AWS services**—Amazon Comprehend is designed to work seamlessly with other AWS services like Amazon S3 and AWS Lambda. Store your documents in Amazon S3, or analyze real-time data with Kinesis Data Firehose. Support for AWS Identity and Access Management (IAM) makes it easy to securely control access to Amazon Comprehend operations. Using IAM, you can create and manage AWS users and groups to grant the appropriate access to your developers and end users.

- **Low cost**—With Amazon Comprehend, you only pay for the documents that you analyze. There are no minimum fees or upfront commitments.

# Are You a First-time User of Amazon Comprehend?

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

1. **How It Works (p. 4)** – This section introduces Amazon Comprehend concepts.
2. **Getting Started with Amazon Comprehend (p. 7)** – In this section, you set up your account and test Amazon Comprehend.
3. **API Reference (p. 92)** – In this section you'll find reference documentation for Amazon Comprehend operations.

# How It Works

Amazon Comprehend uses a pre-trained model to examine and analyze a document or set of documents to gather insights about it. This model is continuously trained on a large body of text so that there is no need for you to provide training data.

Amazon Comprehend can examine and analyze documents in these languages:

- English
- Spanish
- French
- German
- Italian
- Portuguese

Additionally, Amazon Comprehend's Detect the Dominant Language (p. 48) operation can examine documents and determine the dominant language out of a far wider variety of different languages. For more information, see Languages Supported in Amazon Comprehend (p. 4).

With Amazon Comprehend, you can perform the following on your documents:

- Detect the Dominant Language (p. 48)—Examine text to determine the dominant language.
- Detect Entities (p. 50)—Detect textual references to the names of people, places, and items as well as references to dates and quantities.
- Locate Key Phrases (p. 51)—Find key phrases such as "good morning" in a document or set of documents.
- Determine the Sentiment (p. 52)—Analyze documents and determine the dominant sentiment of the text.
- Analyze Syntax (p. 53)—Parse the words in your text and show the speech syntax for each word and enable you to understand the content of the document.
- Topic Modeling (p. 55)—Search the content of documents to determine common themes and topics.

Each operation can be processed in several ways:

- Single-Document Processing (p. 59)—You call Amazon Comprehend with a single document and receive a synchronous response.
- Multiple Document Synchronous Processing (p. 63)—You call Amazon Comprehend with a collection of up to 25 documents and receive a synchronous response.
- Asynchronous Batch Processing (p. 59)—You put a collection of documents into an Amazon S3 bucket and start an asynchronous operation to analyze the documents. The results of the analysis are returned in an S3 bucket.

# Languages Supported in Amazon Comprehend

Amazon Comprehend (except the **Detect Dominant Language** feature) can examine and analyze documents in these languages:

| Code | Language |
|---|---|
| de | German |
| en | English |
| es | Spanish |
| fr | French |
| it | Italian |
| pt | Portuguese |

**Note**
Amazon Comprehend identifies the language using identifiers from RFC 5646 — if there is a 2-letter ISO 639-1 identifier, with a regional subtag if necessary, it uses that. Otherwise, it uses the ISO 639-2 3-letter code. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

Amazon Comprehend's Detect the Dominant Language (p. 48) feature can detect the following languages

| Code | Language | Code | Language | Code | Language |
|---|---|---|---|---|---|
| af | Afrikaans | hy | Armenian | ps | Pushto |
| am | Amharic | ilo | Iloko | qu | Quechua |
| ar | Arabic | id | Indonesian | ro | Romanian |
| as | Assamese | is | Icelandic | ru | Russian |
| az | Azerbaijani | it | Italian | sa | Sanskrit |
| ba | Bashkir | jv | Javanese | si | Sinhala |
| be | Belarusian | ja | Japanese | sk | Slovak |
| bn | Bengali | kn | Kannada | sl | Slovenian |
| bs | Bosnian | ka | Georgian | sd | Sindhi |
| bg | Bulgarian | kk | Kazakh | so | Somali |
| ca | Catalan | km | Central Khmer | es | Spanish |
| ceb | Cebuano | ky | Kirghiz | sq | Albanian |
| cs | Czech | ko | Korean | sr | Serbian |
| cv | Chuvash | ku | Kurdish | su | Sundanese |
| cy | Welsh | la | Latin | sw | Swahili |
| da | Danish | lv | Latvian | sv | Swedish |
| de | German | lt | Lithuanian | ta | Tamil |
| el | Greek | lb | Luxembourgish | tt | Tatar |

| Code | Language | Code | Language | Code | Language |
|---|---|---|---|---|---|
| en | English | ml | Malayalam | te | Telugu |
| eo | Esperanto | mr | Marathi | tg | Tajik |
| et | Estonian | mk | Macedonian | tl | Tagalog |
| eu | Basque | mg | Malagasy | th | Thai |
| fa | Persian | mn | Mongolian | tk | Turkmen |
| fi | Finnish | ms | Malay | tr | Turkish |
| fr | French | my | Burmese | ug | Uighur |
| gd | Scottish Gaelic | ne | Nepali | uk | Ukrainian |
| ga | Irish | new | Newari | ur | Urdu |
| gl | Galician | nl | Dutch | uz | Uzbek |
| gu | Gujarati | no | Norwegian | vi | Vietnamese |
| ht | Haitian | or | Oriya | yi | Yiddish |
| he | Hebrew | pa | Punjabi | yo | Yoruba |
| hi | Hindi | pl | Polish | zh | Chinese (Simplified) |
| hr | Croatian | pt | Portuguese | zh-TW | Chinese (Traditional) |
| hu | Hungarian | | | | |

# Getting Started with Amazon Comprehend

To get started using Amazon Comprehend, 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 Comprehend for the first time, complete the following tasks:

1. Sign Up for AWS (p. 7)
2. Create an IAM User (p. 8)

### 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 Comprehend. You are charged only for the services that you use.

With Amazon Comprehend, you pay only for the resources that you use. If you are a new AWS customer, you can get started with Amazon Comprehend 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://aws.amazon.com/, and then choose **Create an AWS Account**.

   **Note**
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

2. Follow the online instructions.

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

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

## Create an IAM User

Services in AWS, such as Amazon Comprehend, 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.

The Getting Started exercises in this guide assume that you have a user with administrator privileges, `adminuser`.

**To create an administrator user and sign in to the console**

1. Create an administrator user called `adminuser` in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. Sign in to the AWS Management Console using a special URL. For more information, see How Users Sign In to Your Account 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. 8)

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

You don't need the AWS CLI to perform the steps in the Getting Started exercises. However, some of the other exercises in this guide do require it. If you prefer, you can skip this step and go to Step 3: Getting Started Using the Amazon Comprehend Console (p. 9), and set up the AWS CLI later.

**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 the 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 help
```

## Next Step

Step 3: Getting Started Using the Amazon Comprehend Console (p. 9)

# Step 3: Getting Started Using the Amazon Comprehend Console

The easiest way to get started using Amazon Comprehend is to use the console to analyze a short text file. If you haven't reviewed the concepts and terminology in How It Works (p. 4), we recommend that you do that before proceeding.

**Topics**

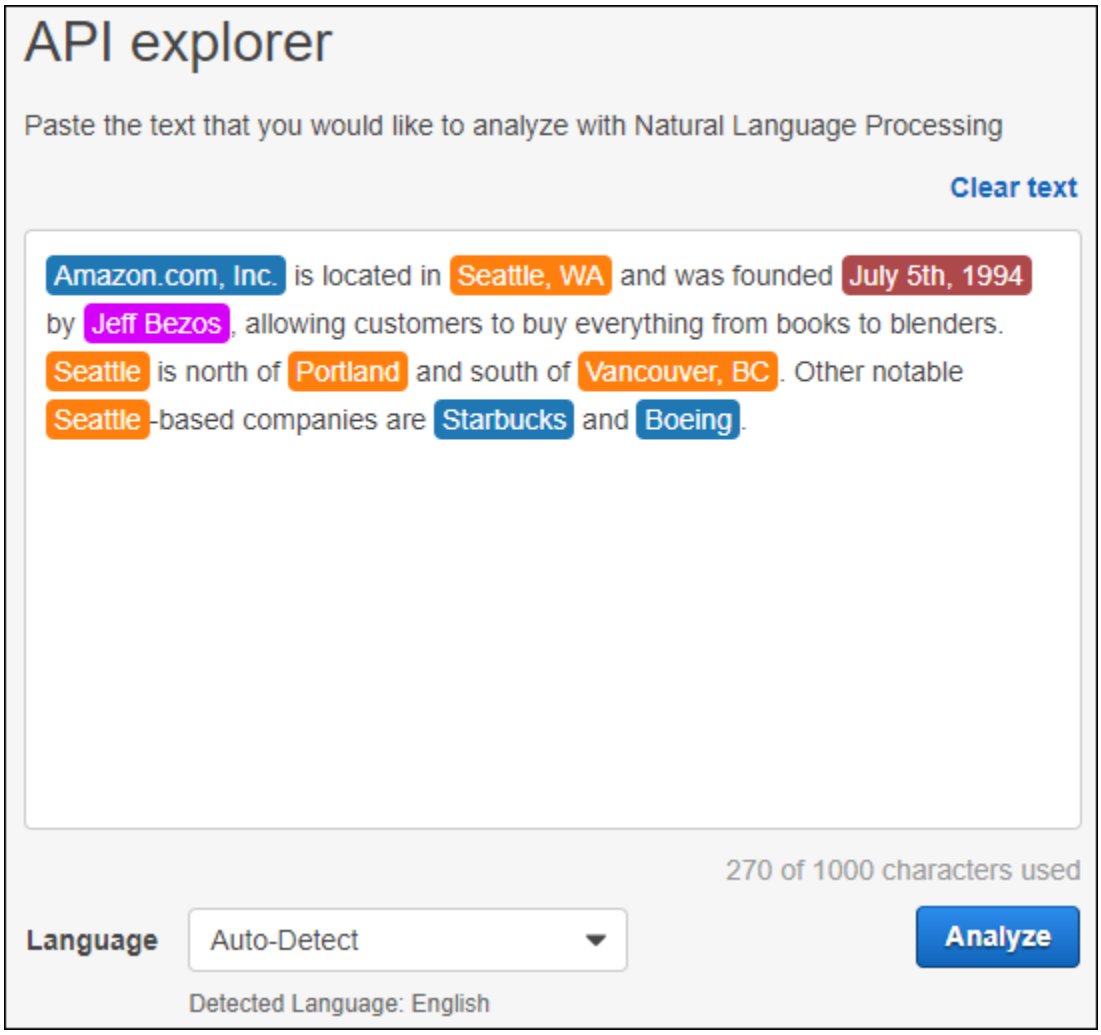


## Analyzing Documents Using the Console

The Amazon Comprehend console enables you to analyze the contents of documents up to 1,000 characters long. The results are shown in the console so that you can review the analysis.

To start analyzing documents, sign in to the AWS Management Console and open the Amazon Comprehend console.

The console displays sample text and the analysis of that text:




You can replace the sample text with your own text in English or Spanish and then choose **Analyze** to get an analysis of your text.

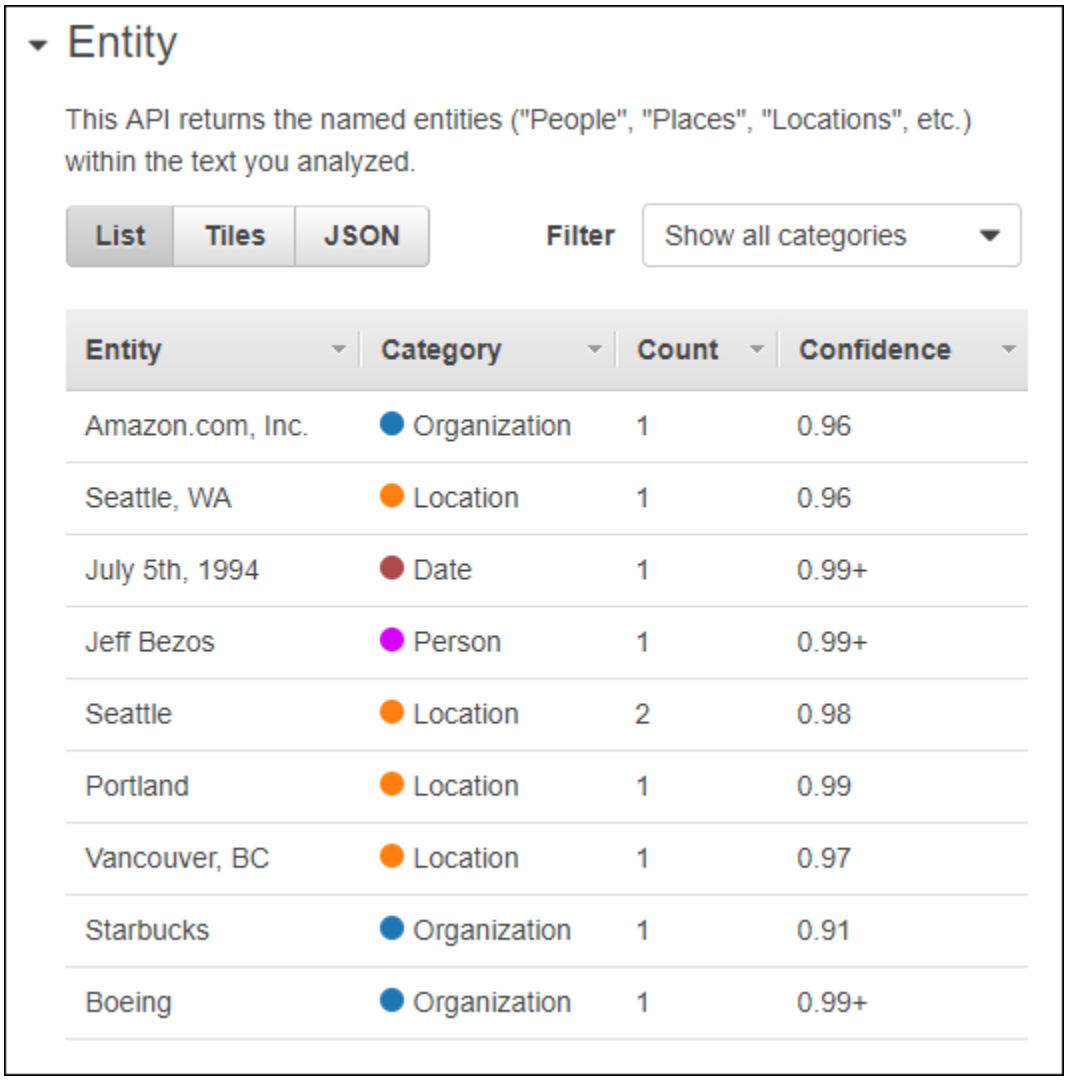
The text is color-coded to indicate the entity type of significant words:

- Orange tags identify locations.
- Brown tags identify dates.
- Magenta tags identify persons.
- Blue tags identify organizations.
- Black tags identify other entities that don't fit into any of the other entity categories.

For more information, see Detect Entities (p. 50)

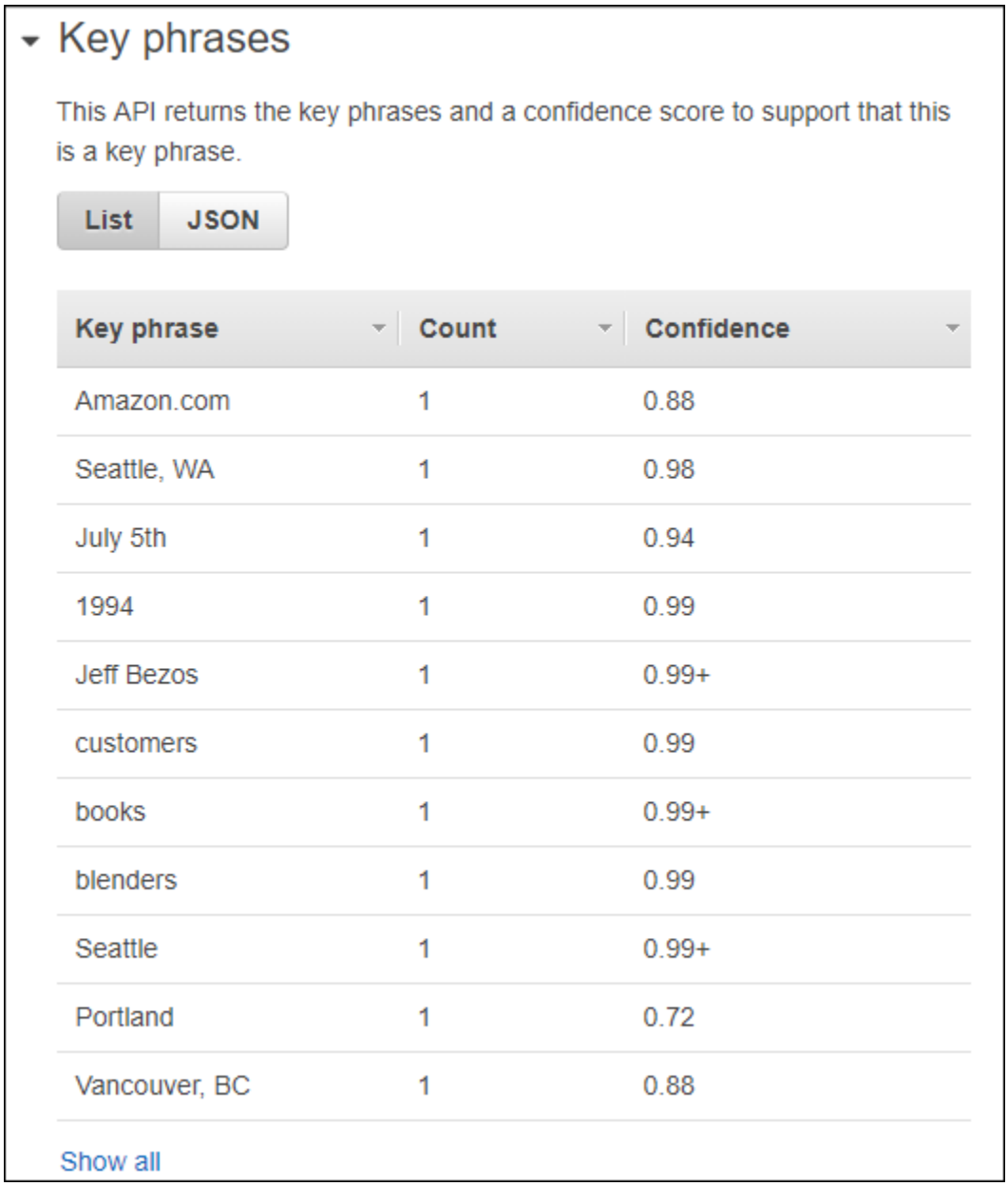
On the right side of the console, the **Analysis** pane shows more information about the text.

The **Entity** section displays cards for the entities found in the text:




Each card shows the text and its entity type. To see a list of all of the entities in the text, choose **List**. For a JSON structure of the results, choose **JSON**. The JSON structure is the same as the structure returned by the DetectEntities (p. 139) operation.

The **Key phrases** section of the **Analysis** pane lists key noun phrases that Amazon Comprehend detected in the input text. For the sample input text, the **Key phrases** section looks like this:



## Key phrases

This API returns the key phrases and a confidence score to support that this is a key phrase.

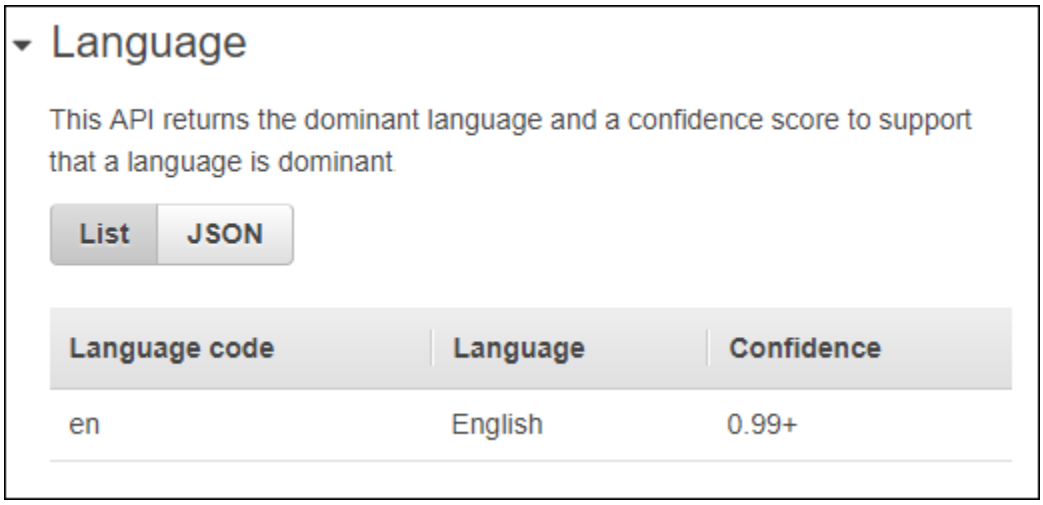
List | JSON

| Key phrase | Count | Confidence |
|---|---|---|
| Amazon.com | 1 | 0.88 |
| Seattle, WA | 1 | 0.98 |
| July 5th | 1 | 0.94 |
| 1994 | 1 | 0.99 |
| Jeff Bezos | 1 | 0.99+ |
| customers | 1 | 0.99 |
| books | 1 | 0.99+ |
| blenders | 1 | 0.99 |
| Seattle | 1 | 0.99+ |
| Portland | 1 | 0.72 |
| Vancouver, BC | 1 | 0.88 |

Show all

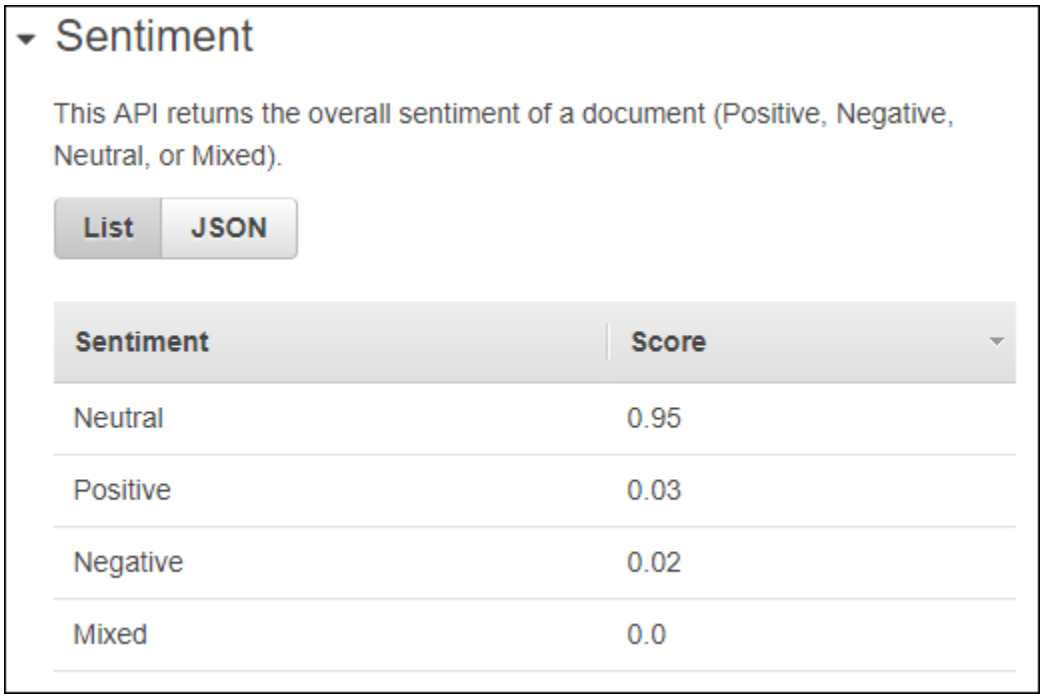
For an alternative view of the results, choose **List** or **JSON** . The JSON structure is the same as the one returned by the DetectKeyPhrases (p. 142) operation.

The **Language** section shows the dominant language for the sample text and the Confidence score. The Confidence score represents the level of confidence that Amazon Comprehend has that it's detected the dominant language correctly. Amazon Comprehend can recognize 100 languages. For more information, see Detect the Dominant Language (p. 48).




As with the other sections, you can choose **List** or **JSON** to get another view of the results. The JSON structure is the same as the one returned by the DetectDominantLanguage (p. 136) operation.

The **Sentiment** section of the **Analysis** pane shows the overall emotional sentiment of the text.




The score represents the confidence that Amazon Comprehend has that it has correctly detected the emotional sentiment expressed in the text. Sentiment can be rated positive, neutral, mixed, or negative.

# Creating a Topic Modeling Job Using the Console

You can use the Amazon Comprehend console to create and manage asynchronous topic detection jobs.

**To create a topic detection job**

1. Sign in to the AWS Management Console and open the Amazon Comprehend console.

2. From the left menu, choose **Organization** and then choose **Create**.
3. Choose the data source to use. You can use either sample data or you can analyze your own data stored in an Amazon S3 bucket. If you use the sample dataset, the topic modeling job analyzes text from this collection of articles: https://s3.amazonaws.com/public-sample-attributions/Attributions.txt
4. If you chose to use your own data, provide the following information in the **Choose input data** section:
   - **S3 data location** – An Amazon S3 data bucket that contains the documents to analyze. You can choose the folder icon to browse to the location of your data. The bucket must be in the same region as the API that you are calling.
   - **Input format** – Choose whether input data is contained in one document per file, or if there is one document per line in a file.
   - **Number of topics** – The number of topics to return.
   - **Job name** – A name that identifies the particular job.
5. In the **Choose output location** section, provide the following:
   - **S3 data location** – an Amazon S3 data bucket where the results of the analysis will be written. The bucket must be in the same region as the API that your are calling.
6. In the **Choose an IAM role** section, either select an existing IAM role or create a new one.
   - **Choose an existing IAM role** – Choose this option if you already have an IAM role with permissions to access the input and output Amazon S3 buckets.
   - **Create a new IAM role** – Choose this option when you want to create a new IAM role with the proper permissions for Amazon Comprehend to access the input and output buckets. For more information about the permissions given to the IAM role, see Role-Based Permissions Required for Asynchronous Operations (p. 85).
7. When you have finished filling out the form, choose **Create job** to create and start the topic detection job.

## Select input data

Please select the topic modeling data you would like to analyze. Our text a
1,000 documents of at least 100 words each, but can be applied to docum
millions of documents.

- (•) Topic modeling sample data
- ( ) My data (S3)

**S3 data location** s3://public-sample-us-west-2

**Input format** One document per line

**Number of topics** 10

**Job Name** <job name>

Any characters; length between 1-256

## Select output location

Select the preferred output format for your analysis. S3 data output locatio

**S3 data location** s3://<output bucket>

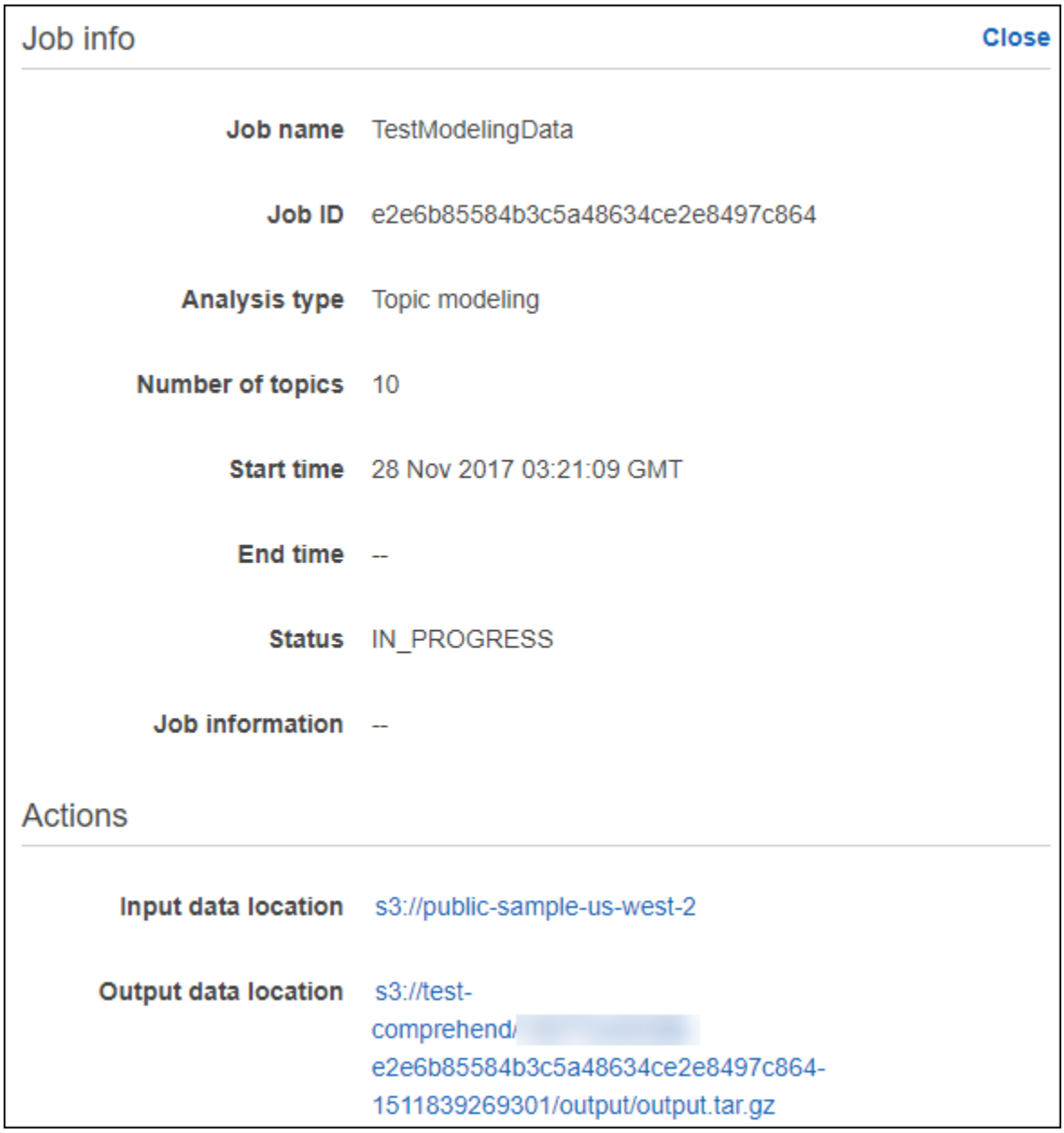
## Select an IAM role

The topic modeling job will use the IAM role to access your Amazon S3 inp

- ( ) Select an existing IAM role
- (•) Create a new IAM role

**Name suffix** <IAM role name>

The new job appears in the job list with the status field showing the status of the job. The field can be `IN_PROGRESS` for a job that is processing, `COMPLETED` for a job that has finished successfully, and `FAILED` for a job that has an error. You can click on a job to get more information about the job, including any error messages.




**Next Step**

Step 4: Getting Started Using the Amazon Comprehend API (p. 17)

# Step 4: Getting Started Using the Amazon Comprehend API

The following examples demonstrate how to use Amazon Comprehend operations using the AWS CLI, Java, and Python. Use them to learn about Amazon Comprehend operations and as building blocks for your own applications.

To run the AWS CLI and Python examples, you need to install the AWS CLI. For more information, see Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 8).

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**



## Detecting the Dominant Language

To determine the dominant language used in text, use the Amazon Comprehend DetectDominantLanguage (p. 136) operation. To detect the dominant language in up to 25 documents in a batch, use the BatchDetectDominantLanguage (p. 94) operation. For more information, see Using the Batch APIs (p. 42).

**Topics**



### Detecting the Dominant Language Using the AWS Command Line Interface

The following example demonstrates using the `DetectDominantLanguage` operation with the AWS CLI.

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 (^).

```
aws comprehend detect-dominant-language \
    --region region \
```

```
    --text "It is raining today in Seattle."
```

Amazon Comprehend responds with the following:

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9793661236763
        }
    ]
}
```

## Detecting the Dominant Language Using the AWS SDK for Java

The following example uses the `DetectDominantLanguage` operation with Java.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectDominantLanguageRequest;
import com.amazonaws.services.comprehend.model.DetectDominantLanguageResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // 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();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectDominantLanguage API
        System.out.println("Calling DetectDominantLanguage");
        DetectDominantLanguageRequest detectDominantLanguageRequest = new
 DetectDominantLanguageRequest().withText(text);
        DetectDominantLanguageResult detectDominantLanguageResult =
 comprehendClient.detectDominantLanguage(detectDominantLanguageRequest);
        detectDominantLanguageResult.getLanguages().forEach(System.out::println);
        System.out.println("Calling DetectDominantLanguage\n");
        System.out.println("Done");
    }
}
```

## Detecting the Dominant Language Using the AWS SDK for Python (Boto)

The following example demonstrates using the `DetectDominantLanguage` operation with Python.

```
import boto3
import json
```

```
comprehend = boto3.client(service_name='comprehend', region_name='region')
text = "It is raining today in Seattle"

print('Calling DetectDominantLanguage')
print(json.dumps(comprehend.detect_dominant_language(Text = text), sort_keys=True,
 indent=4))
print("End of DetectDominantLanguage\n")
```

## Detecting the Dominant Language Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectDominantLanguage API
            Console.WriteLine("Calling DetectDominantLanguage\n");
            DetectDominantLanguageRequest detectDominantLanguageRequest = new
 DetectDominantLanguageRequest()
            {
                Text = text
            };
            DetectDominantLanguageResponse detectDominantLanguageResponse =
 comprehendClient.DetectDominantLanguage(detectDominantLanguageRequest);
            foreach (DominantLanguage dl in detectDominantLanguageResponse.Languages)
                Console.WriteLine("Language Code: {0}, Score: {1}", dl.LanguageCode,
 dl.Score);
            Console.WriteLine("Done");
        }
    }
}
```

# Detecting Named Entities

To determine the named entities in a document, use the Amazon Comprehend DetectEntities (p. 139) operation. To detect entities in up to 25 documents in a batch, use the BatchDetectEntities (p. 97) operation. For more information, see Using the Batch APIs (p. 42).

**Topics**

## Detecting Named Entities Using the AWS Command Line Interface

The following example demonstrates using the `DetectEntities` operation using the AWS CLI. You must specify the language of the input 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 (^).

```
aws comprehend detect-entities \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend responds with the following:

```
{
    "Entities": [
        {
            "Text": "today",
            "Score": 0.97,
            "Type": "DATE",
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.95,
            "Type": "LOCATION",
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ],
    "LanguageCode": "en"
}
```

## Detecting Named Entities Using the AWS SDK for Java

The following example uses the `DetectEntities` operation with Java. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectEntitiesRequest;
import com.amazonaws.services.comprehend.model.DetectEntitiesResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // 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();
```

```
        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectEntities API
        System.out.println("Calling DetectEntities");
        DetectEntitiesRequest detectEntitiesRequest = new
 DetectEntitiesRequest().withText(text)

 .withLanguageCode("en");
        DetectEntitiesResult detectEntitiesResult  =
 comprehendClient.detectEntities(detectEntitiesRequest);
        detectEntitiesResult.getEntities().forEach(System.out::println);
        System.out.println("End of DetectEntities\n");
    }
}
```

## Detecting Named Entities Using the AWS SDK for Python (Boto)

The following example uses the `DetectEntities` operation with Python. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')
text = "It is raining today in Seattle"

print('Calling DetectEntities')
print(json.dumps(comprehend.detect_entities(Text=text, LanguageCode='en'), sort_keys=True,
 indent=4))
print('End of DetectEntities\n')
```

## Detecting Entities Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectEntities API
            Console.WriteLine("Calling DetectEntities\n");
            DetectEntitiesRequest detectEntitiesRequest =  new DetectEntitiesRequest()
            {
```

```
                Text = text,
                LanguageCode = "en"
            };
            DetectEntitiesResponse detectEntitiesResponse =
 comprehendClient.DetectEntities(detectEntitiesRequest);
            foreach (Entity e in detectEntitiesResponse.Entities)
                Console.WriteLine("Text: {1}, Type: {1}, Score: {2}, BeginOffset: {3},
 EndOffset: {4}",
                        e.Text, e.Type, e.Score, e.BeginOffset, e.EndOffset);
            Console.WriteLine("Done");
        }
    }
}
```

# Detecting Key Phrases

To determine the key noun phrases used in text, use the Amazon Comprehend DetectKeyPhrases (p. 142) operation. To detect the key noun phrases in up to 25 documents in a batch, use the BatchDetectKeyPhrases (p. 100) operation. For more information, see Using the Batch APIs (p. 42).

**Topics**

- Detecting Key Phrases Using the AWS Command Line Interface (p. 22)
- Detecting Key Phrases Using the AWS SDK for Java (p. 23)
- Detecting Key Phrases Using the AWS SDK for Python (Boto) (p. 23)
- Detecting Key Phrases Using the AWS SDK for .NET (p. 24)

## Detecting Key Phrases Using the AWS Command Line Interface

The following example demonstrates using the `DetectKeyPhrases` operation with the AWS CLI. You must specify the language of the input 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 (^).

```
aws comprehend detect-key-phrases \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend responds with the following:

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
            "BeginOffset": 23,
            "EndOffset": 30
        }
```

```
    ]
}
```

## Detecting Key Phrases Using the AWS SDK for Java

The following example uses the `DetectKeyPhrases` operation with Java. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectKeyPhrasesRequest;
import com.amazonaws.services.comprehend.model.DetectKeyPhrasesResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // 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();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectKeyPhrases API
        System.out.println("Calling DetectKeyPhrases");
        DetectKeyPhrasesRequest detectKeyPhrasesRequest = new
 DetectKeyPhrasesRequest().withText(text)

 .withLanguageCode("en");
        DetectKeyPhrasesResult detectKeyPhrasesResult =
 comprehendClient.detectKeyPhrases(detectKeyPhrasesRequest);
        detectKeyPhrasesResult.getKeyPhrases().forEach(System.out::println);
        System.out.println("End of DetectKeyPhrases\n");
    }
}
```

## Detecting Key Phrases Using the AWS SDK for Python (Boto)

The following example uses the `DetectKeyPhrases` operation with Python. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')

text = "It is raining today in Seattle"

print('Calling DetectKeyPhrases')
print(json.dumps(comprehend.detect_key_phrases(Text=text, LanguageCode='en'),
 sort_keys=True, indent=4))
print('End of DetectKeyPhrases\n')
```

## Detecting Key Phrases Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectKeyPhrases");
            DetectKeyPhrasesRequest detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
            {
                Text = text,
                LanguageCode = "en"
            };
            DetectKeyPhrasesResponse detectKeyPhrasesResponse =
 comprehendClient.DetectKeyPhrases(detectKeyPhrasesRequest);
            foreach (KeyPhrase kp in detectKeyPhrasesResponse.KeyPhrases)
                Console.WriteLine("Text: {1}, Type: {1}, BeginOffset: {2}, EndOffset: {3}",
                    kp.Text, kp.Text, kp.BeginOffset, kp.EndOffset);
            Console.WriteLine("Done");
        }
    }
}
```

# Detecting Sentiment

To determine the overall emotional tone of text, use the DetectSentiment (p. 145) operation. To detect the sentiment in up to 25 documents in a batch, use the BatchDetectSentiment (p. 103) operation. For more information, see Using the Batch APIs (p. 42).

**Topics**



## Detecting Sentiment Using the AWS Command Line Interface

The following example demonstrates using the `DetectSentiment` operation with the AWS CLI. This example specifies the language of the input 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 (^).

```
aws comprehend detect-sentiment \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend responds with the following:

```
{
    "SentimentScore": {
        "Mixed": 0.014585512690246105,
        "Positive": 0.31592071056365967,
        "Neutral": 0.5985543131828308,
        "Negative": 0.07093945890665054
    },
    "Sentiment": "NEUTRAL",
    "LanguageCode": "en"
}
```

## Detecting Sentiment Using the AWS SDK for Java

The following example Java program detects the sentiment of input text. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectSentimentRequest;
import com.amazonaws.services.comprehend.model.DetectSentimentResult;

public class App
{
    public static void main( String[] args )
    {

        String text = "It is raining today in Seattle";

        // 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();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        // Call detectSentiment API
        System.out.println("Calling DetectSentiment");
        DetectSentimentRequest detectSentimentRequest = new
 DetectSentimentRequest().withText(text)

 .withLanguageCode("en");
        DetectSentimentResult detectSentimentResult =
 comprehendClient.detectSentiment(detectSentimentRequest);
        System.out.println(detectSentimentResult);
        System.out.println("End of DetectSentiment\n");
        System.out.println( "Done" );
    }
}
```

## Detecting Sentiment Using the AWS SDK for Python (Boto)

The following Python program detects the sentiment of input text. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')

text = "It is raining today in Seattle"

print('Calling DetectSentiment')
print(json.dumps(comprehend.detect_sentiment(Text=text, LanguageCode='en'), sort_keys=True,
 indent=4))
print('End of DetectSentiment\n')
```

## Detecting Sentiment Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectSentiment");
            DetectSentimentRequest detectSentimentRequest = new DetectSentimentRequest()
            {
                Text = text,
                LanguageCode = "en"
            };
            DetectSentimentResponse detectSentimentResponse =
 comprehendClient.DetectSentiment(detectSentimentRequest);
            Console.WriteLine(detectSentimentResponse.Sentiment);
            Console.WriteLine("Done");
        }
    }
}
```

# Detecting Syntax

To parse text to extract the individual words and determine the parts of speech for each word, use the DetectSyntax (p. 148) operation. To parse the syntax of up to 25 documents in a batch, use the BatchDetectSyntax (p. 106) operation. For more information, see Using the Batch APIs (p. 42).

**Topics**



# Detecting Syntax Using the AWS Command Line Interface.

The following example demonstrates using the `DetectSyntax` operation with the AWS CLI. This example specifies the language of the input 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 (^).

```
aws comprehend detect-syntax \
   --region region \
   --language-code "en" \
   --text "It is raining today in Seattle."
```

Amazon Comprehend responds with the following:

```
{
    "SyntaxTokens": [
        {
            "Text": "It",
            "EndOffset": 2,
            "BeginOffset": 0,
            "PartOfSpeech": {
                "Tag": "PRON",
                "Score": 0.8389829397201538
            },
            "TokenId": 1
        },
        {
            "Text": "is",
            "EndOffset": 5,
            "BeginOffset": 3,
            "PartOfSpeech": {
                "Tag": "AUX",
                "Score": 0.9189288020133972
            },
            "TokenId": 2
        },
        {
            "Text": "raining",
            "EndOffset": 13,
            "BeginOffset": 6,
            "PartOfSpeech": {
                "Tag": "VERB",
                "Score": 0.9977611303329468
            },
            "TokenId": 3
        },
        {
            "Text": "today",
            "EndOffset": 19,
            "BeginOffset": 14,
            "PartOfSpeech": {
                "Tag": "NOUN",
                "Score": 0.9993606209754944
```

```
            },
            "TokenId": 4
        },
        {
            "Text": "in",
            "EndOffset": 22,
            "BeginOffset": 20,
            "PartOfSpeech": {
                "Tag": "ADP",
                "Score": 0.9999061822891235
            },
            "TokenId": 5
        },
        {
            "Text": "Seattle",
            "EndOffset": 30,
            "BeginOffset": 23,
            "PartOfSpeech": {
                "Tag": "PROPN",
                "Score": 0.9940338730812073
            },
            "TokenId": 6
        },
        {
            "Text": ".",
            "EndOffset": 31,
            "BeginOffset": 30,
            "PartOfSpeech": {
                "Tag": "PUNCT",
                "Score": 0.9999997615814209
            },
            "TokenId": 7
        }
    ]
}
```

## Detecting Syntax Using the AWS SDK for Java

The following Java program detects the syntax of the input text. You must specify the language of the input text.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DetectSyntaxRequest;
import com.amazonaws.services.comprehend.model.DetectSyntaxResult;

public class App
{
 public static void main( String[] args )
 {

  String text = "It is raining today in Seattle.";
  String region = "region"

  // 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();

  AmazonComprehend comprehendClient =
    AmazonComprehendClientBuilder.standard()
      .withCredentials(awsCreds)
      .withRegion(region)
```

```
      .build();

  // Call detectSyntax API
  System.out.println("Calling DetectSyntax");
  DetectSyntaxRequest detectSyntaxRequest = new DetectSyntaxRequest()
    .withText(text)
    .withLanguageCode("en");
  DetectSyntaxResult detectSyntaxResult =
 comprehendClient.detectSyntax(detectSyntaxRequest);
  detectSyntaxResult.getSyntaxTokens().forEach(System.out::println);
  System.out.println("End of DetectSyntax\n");
  System.out.println( "Done" );
 }
}
```

## Detecting Parts of Speech Using the AWS SDK for Python (Boto)

The following Python program detects the parts of speech in the input text. You must specify the language of the input text.

```
import boto3
import json

comprehend = boto3.client(service_name='comprehend', region_name='region')
text = "It is raining today in Seattle"

print('Calling DetectSyntax')
print(json.dumps(comprehend.detect_syntax(Text=text, LanguageCode='en'), sort_keys=True,
 indent=4))
print('End of DetectSyntax\n')
```

## Detecting Syntax Using the AWS SDK for .NET

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
 class Program
 {
  static void Main(string[] args)
  {
   String text = "It is raining today in Seattle";

   AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.region);

   // Call DetectSyntax API
   Console.WriteLine("Calling DetectSyntax\n");
   DetectSyntaxRequest detectSyntaxRequest =  new DetectSyntaxRequest()
   {
    Text = text,
    LanguageCode = "en"
   };
   DetectSyntaxResponse detectSyntaxResponse =
 comprehendClient.DetectSyntax(detectSyntaxRequest);
```

```
   foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
   Console.WriteLine("Text: {0}, PartOfSpeech: {1}, Score: {2}, BeginOffset: {3},
 EndOffset: {4}",
     e.Text, e.PartOfSpeech, e.Score, e.BeginOffset, e.EndOffset);
   Console.WriteLine("Done");
  }
 }
}
```

# Using Custom Classification

To create and train a custom classifier, use the Amazon Comprehend the section called "CreateDocumentClassifier" (p. 109). To identify custom classifiers in a corpus of documents, use the the section called "StartDocumentClassificationJob" (p. 174) operation.

**Topics**



## Using Custom Classification Using the AWS Command Line Interface

The following examples demonstrates using the `CreateDocumentClassifier` operation, `StartDocumentClassificationJob` operation, and other custom classifier APIs with the AWS CLI.

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 (^).

Creating a custom classifier using the `create-document-classifier` operation.

```
aws comprehend create-document-classifier \
     --region region \
     --document-classifier-name testDelete \
     --language-code en \
     --input-data-config S3Uri=s3://S3Bucket/docclass/file name \
     --data-access-role-arn arn:aws:iam::account number:role/testDeepInsightDataAccess
```

Getting information on a custom classifier with the document classifier arn using the `DescribeDocumentClassifier` operation.

```
aws comprehend describe-document-classifier \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/file name
```

Deleting a custom classifier using the `DeleteDocumentClassifier` operation.

```
aws comprehend delete-document-classifier \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/testDelete
```

List all custom classifiers in the account using the `ListDocumentClassifiers` operation.

```
aws comprehend list-document-classifiers
```

```
    --region region
```

Run a custom classification job using the `StartDocumentClassificationJob` operation.

```
aws comprehend start-document-classification-job \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/testDelete \
     --input-data-config S3Uri=s3://S3Bucket/docclass/file
 name,InputFormat=ONE_DOC_PER_LINE \
     --output-data-config S3Uri=s3://S3Bucket/output \
     --data-access-role-arn arn:aws:iam::account number:role/resource name
```

Getting information on a custom classifier with the job id using the `DescribeDocumentClassificationJob` operation.

```
aws comprehend describe-document-classification-job \
     --region region \
     --job-id job id
```

Listing all custom classification jobs in your account using the `ListDocumentClassificationJobs` operation.

```
aws comprehend list-document-classification-jobs
     --region region
```

## Using Custom Classification Using the AWS SDK for Java

This example creates a custom classifier and trains it using Java

```
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.CreateDocumentClassifierRequest;
import com.amazonaws.services.comprehend.model.CreateDocumentClassifierResult;
import com.amazonaws.services.comprehend.model.DescribeDocumentClassifierRequest;
import com.amazonaws.services.comprehend.model.DescribeDocumentClassifierResult;
import com.amazonaws.services.comprehend.model.DocumentClassifierInputDataConfig;
import com.amazonaws.services.comprehend.model.LanguageCode;
import com.amazonaws.services.comprehend.model.ListDocumentClassifiersRequest;
import com.amazonaws.services.comprehend.model.ListDocumentClassifiersResult;

public class DocumentClassifierDemo {

    public static void main(String[] args) {
        final AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withRegion("us-west-2")
                                         .build();

        final String dataAccessRoleArn = "arn:aws:iam::account number:role/resource name";

        final CreateDocumentClassifierRequest createDocumentClassifierRequest = new
 CreateDocumentClassifierRequest()
            .withDocumentClassifierName("SampleCodeClassifier")
            .withDataAccessRoleArn(dataAccessRoleArn)
            .withLanguageCode(LanguageCode.En)
            .withInputDataConfig(new DocumentClassifierInputDataConfig()
                .withS3Uri("s3://S3Bucket/docclass/file name"));

        final CreateDocumentClassifierResult createDocumentClassifierResult =
            comprehendClient.createDocumentClassifier(createDocumentClassifierRequest);
```

```
        final String documentClassifierArn =
 createDocumentClassifierResult.getDocumentClassifierArn();

        System.out.println("Document Classifier ARN: " + documentClassifierArn);

        final DescribeDocumentClassifierRequest describeDocumentClassifierRequest = new
 DescribeDocumentClassifierRequest()
            .withDocumentClassifierArn(documentClassifierArn);
        final DescribeDocumentClassifierResult describeDocumentClassifierResult =
 comprehendClient.describeDocumentClassifier(describeDocumentClassifierRequest);
        System.out.println("DescribeDocumentClassifierResult: " +
 describeDocumentClassifierResult);

        final ListDocumentClassifiersRequest listDocumentClassifiersRequest = new
 ListDocumentClassifiersRequest();

        final ListDocumentClassifiersResult listDocumentClassifiersResult =
 comprehendClient
            .listDocumentClassifiers(listDocumentClassifiersRequest);
        System.out.println("ListDocumentClassifierResult: " +
 listDocumentClassifiersResult );
    }
}
```

## Using Custom Classification Using the AWS SDK for Python (Boto)

This example creates a custom classifier and trains it using Python

```
import boto3

# Instantiate Boto3 SDK:
client = boto3.client('comprehend', region_name='region')

# Create a document classifier
create_response = client.create_document_classifier(
    InputDataConfig={
        'S3Uri': 's3://S3Bucket/docclass/file name'
    },
    DataAccessRoleArn='arn:aws:iam::account number:role/resource name',
    DocumentClassifierName='SampleCodeClassifier1',
    LanguageCode='en'
)
print("Create response: %s\n", create_response)

# Check the status of the classifier
describe_response = client.describe_document_classifier(
    DocumentClassifierArn=create_response['DocumentClassifierArn'])
print("Describe response: %s\n", describe_response)

# List all classifiers in account
list_response = client.list_document_classifiers()
print("List response: %s\n", list_response)
```

This example runs a custom classifier job using Python

```
import boto3

# Instantiate Boto3 SDK:
client = boto3.client('comprehend', region_name='region')

start_response = client.start_document_classification_job(
```

```
    InputDataConfig={
        'S3Uri': 's3://srikad-us-west-2-input/docclass/file name',
        'InputFormat': 'ONE_DOC_PER_LINE'
    },
    OutputDataConfig={
        'S3Uri': 's3://S3Bucket/output'
    },
    DataAccessRoleArn='arn:aws:iam::account number:role/resource name',
    DocumentClassifierArn=
    'arn:aws:comprehend:region:account number:document-classifier/SampleCodeClassifier1'
)

print("Start response: %s\n", start_response)

# Check the status of the job
describe_response =
 client.describe_document_classification_job(JobId=start_response['JobId'])
print("Describe response: %s\n", describe_response)

# List all classification jobs in account
list_response = client.list_document_classification_jobs()
print("List response: %s\n", list_response)
```

# Detecting Custom Entities

To create the custom entities in a document, use the Amazon Comprehend CreateEntityRecognizer (p. 112) to create an entity recognizer. To identify those custom entities, use the StartEntitiesDetectionJob (p. 181) operation.

**Topics**

- Creating and Detecting Custom Entities Using the AWS Command Line Interface (p. 33)
- Detecting Custom Entities Using the AWS SDK for Java (p. 34)
- Detecting Custom Entities Using the AWS Using the AWS SDK for Python (Boto3) (p. 35)

## Creating and Detecting Custom Entities Using the AWS Command Line Interface

The following examples demonstrates using the `CreateEntityRecognizer` operation, `StartEntitiesDetectionJob` operation and other associated APIs with the AWS CLI.

The examples are formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

Creating a custom entity recognizer using the `CreateEntityRecognizer` operation.

```
aws comprehend create-entity-recognizer \
     --language-code en \
     --recognizer-name test-6 \
     --data-access-role-arn "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role" \
     --input-data-config "EntityTypes=[{Type=PERSON}],Documents={S3Uri=s3://Bucket
 Name/Bucket Path/documents},
                 Annotations={S3Uri=s3://Bucket Name/Bucket Path/annotations}" \
     --region region
```

Listing all entity recognizers in a region using the `ListEntityRecognizers` operation.

```
aws comprehend list-entity-recognizers \
```

```
    --region region
```

Checking Job Status of custom entity recognizers using the `DescribeEntityRecognizer` operation.

```
aws comprehend describe-entity-recognizer \
    --entity-recognizer-arn arn:aws:comprehend:region:account number:entity-recognizer/
test-6 \
    --region region
```

Starting a custom entities recogniztion job using the `StartEntitiesDetectionJob` operation.

```
aws comprehend start-entities-detection-job \
    --entity-recognizer-arn "arn:aws:comprehend:region:account number:entity-recognizer/
test-6" \
    --job-name infer-1 \
    --data-access-role-arn "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role" \
    --language-code en \
    --input-data-config "S3Uri=s3://Bucket Name/Bucket Path" \
    --output-data-config "S3Uri=s3://Bucket Name/Bucket Path/" \
    --region region
```

## Detecting Custom Entities Using the AWS SDK for Java

This example creates a custom entity recognizer, trains the model, and then runs it in an entity recognizer job using Java

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.CreateEntityRecognizerRequest;
import com.amazonaws.services.comprehend.model.CreateEntityRecognizerResult;
import com.amazonaws.services.comprehend.model.DescribeEntityRecognizerRequest;
import com.amazonaws.services.comprehend.model.DescribeEntityRecognizerResult;
import com.amazonaws.services.comprehend.model.EntityRecognizerAnnotations;
import com.amazonaws.services.comprehend.model.EntityRecognizerDocuments;
import com.amazonaws.services.comprehend.model.EntityRecognizerInputDataConfig;
import com.amazonaws.services.comprehend.model.EntityTypesListItem;
import com.amazonaws.services.comprehend.model.InputDataConfig;
import com.amazonaws.services.comprehend.model.LanguageCode;
import com.amazonaws.services.comprehend.model.OutputDataConfig;
import com.amazonaws.services.comprehend.model.StartEntitiesDetectionJobRequest;
import com.amazonaws.services.comprehend.model.StartEntitiesDetectionJobResult;

public class CustomEntityRecognizerDemo {

    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();

        AmazonComprehend comprehendClient =
                AmazonComprehendClientBuilder.standard()
                        .withCredentials(awsCreds)
                        .withRegion("region")
                        .build();

        final String dataAccessRoleArn = "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role";
```

```
        final CreateEntityRecognizerRequest createEntityRecognizerRequest = new
 CreateEntityRecognizerRequest()
                .withRecognizerName("recognizer name")
                .withDataAccessRoleArn(dataAccessRoleArn)
                .withLanguageCode(LanguageCode.En)
                .withInputDataConfig(new EntityRecognizerInputDataConfig()
                    .withEntityTypes(new EntityTypesListItem().withType("PERSON"))
                    .withDocuments(new EntityRecognizerDocuments()
                            .withS3Uri("s3://Bucket Name/Bucket Path/documents"))
                    .withAnnotations(new EntityRecognizerAnnotations()
                            .withS3Uri("s3://Bucket Name/Bucket Path/annotations")));

        final CreateEntityRecognizerResult createEntityRecognizerResult =
 comprehendClient.createEntityRecognizer(createEntityRecognizerRequest);
        final String entityRecognizerArn =
 createEntityRecognizerResult.getEntityRecognizerArn();
        System.out.println("Entity Recognizer ARN: " + entityRecognizerArn);

        DescribeEntityRecognizerRequest describeEntityRecognizerRequest = new
 DescribeEntityRecognizerRequest()
                .withEntityRecognizerArn(entityRecognizerArn);
        final DescribeEntityRecognizerResult describeEntityRecognizerResult =
 comprehendClient.describeEntityRecognizer(describeEntityRecognizerRequest);
        System.out.println("describeEntityRecognizerResult: " +
 describeEntityRecognizerResult);

        if
 ("TRAINED".equals(describeEntityRecognizerResult.getEntityRecognizerProperties().getStatus()))
 {
            // After model gets trained, launch an job to extract entities.
            final StartEntitiesDetectionJobRequest startEntitiesDetectionJobRequest = new
 StartEntitiesDetectionJobRequest()
                    .withJobName("Inference Job Name")
                    .withEntityRecognizerArn(entityRecognizerArn)
                    .withDataAccessRoleArn(dataAccessRoleArn)
                    .withLanguageCode(LanguageCode.En)
                    .withInputDataConfig(new InputDataConfig()
                            .withS3Uri("s3://Bucket Name/Bucket Path"))
                    .withOutputDataConfig(new OutputDataConfig()
                            .withS3Uri("s3://Bucket Name/Bucket Path/"));

            final StartEntitiesDetectionJobResult startEntitiesDetectionJobResult =
 comprehendClient.startEntitiesDetectionJob(startEntitiesDetectionJobRequest);
            System.out.println("startEntitiesDetectionJobResult: " +
 startEntitiesDetectionJobResult);
        }
    }

}
```

## Detecting Custom Entities Using the AWS Using the AWS SDK for Python (Boto3)

Instantiate Boto3 SDK:

```
import boto3
import uuid
comprehend = boto3.client("comprehend", region_name="recognizer name")
```

Create entity recognizer:

```
response = comprehend.create_entity_recognizer(
```

```
    RecognizerName="Recognizer-Name-Goes-Here-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "EntityTypes": [
            {
                "Type": "ENTITY_TYPE"
            }
        ],
        "Documents": {
            "S3Uri": "s3://Bucket Name/Bucket Path/documents"
        },
        "Annotations": {
            "S3Uri": "s3://Bucket Name/Bucket Path/annotations"
        }
    }
)
recognizer_arn = response["EntityRecognizerArn"]
```

List all recognizers:

```
response = comprehend.list_entity_recognizers()
```

Wait for recognizer to reach TRAINED status:

```
while True:
    response = comprehend.describe_entity_recognizer(
        EntityRecognizerArn=recognizer_arn
    )

    status = response["EntityRecognizerProperties"]["Status"]
    if "IN_ERROR" == status:
        sys.exit(1)
    if "TRAINED" == status:
        break

    time.sleep(10)
```

Start entities detection job:

```
response = comprehend.start_entities_detection_job(
    EntityRecognizerArn=recognizer_arn,
    JobName="Detection-Job-Name-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "InputFormat": "ONE_DOC_PER_LINE",
        "S3Uri": "s3://Bucket Name/Bucket Path/documents"
    },
    OutputDataConfig={
        "S3Uri": "s3://Bucket Name/Bucket Path/output"
    }
)
```

# Topic Modeling

To determine the topics in a document set, use the StartTopicsDetectionJob (p. 191) to start an asynchronous job. You can monitor topics in documents written in English or Spanish.

**Topics**

## Before You Start

Before you start, make sure that you have:

- **Input and output buckets**—Identify the Amazon S3 buckets that you want to use for input and output. The buckets must be in the same region as the API that you are calling.
- **IAM service role**—You must have an IAM service role with permission to access your input and output buckets. For more information, see Role-Based Permissions Required for Asynchronous Operations (p. 85).

## Topic Modeling Using the AWS Command Line Interface

The following example demonstrates using the `StartTopicsDetectionJob` operation with the AWS CLI

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 (^).

```
aws comprehend start-topics-detection-job \
                --number-of-topics topics to return \
                --job-name "job name" \
                --region region \
                --cli-input-json file://path to JSON input file
```

For the `cli-input-json` parameter you supply the path to a JSON file that contains the request data, as shown in the following example.

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_FILE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
}
```

If the request to start the topic detection job was successful, you will receive the following response:

```
{
    "JobStatus": "SUBMITTED",
    "JobId": "job ID"
}
```

Use the ListTopicsDetectionJobs (p. 171) operation to see a list of the topic detection jobs that you have submitted. The list includes information about the input and output locations that you used as well as the status of each of the detection jobs. 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 (^).

```
aws comprehend list-topics-detection-jobs \
    -- region
```

You will get JSON similar to the following in response:

```
{
    "TopicsDetectionJobPropertiesList": [
        {
            "InputDataConfig": {
                "S3Uri": "s3://input bucket/input path",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "NumberOfTopics": topics to return,
            "JobId": "job ID",
            "JobStatus": "COMPLETED",
            "JobName": "job name",
            "SubmitTime": timestamp,
            "OutputDataConfig": {
                "S3Uri": "s3://output bucket/output path"
            },
            "EndTime": timestamp
        },
        {
            "InputDataConfig": {
                "S3Uri": "s3://input bucket/input path",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "NumberOfTopics": topics to return,
            "JobId": "job ID",
            "JobStatus": "RUNNING",
            "JobName": "job name",
            "SubmitTime": timestamp,
            "OutputDataConfig": {
                "S3Uri": "s3://output bucket/output path"
            }
        }
    ]
}
```

You can use the DescribeTopicsDetectionJob (p. 134) operation to get the status of an existing job. 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 (^).

```
aws comprehend describe-topics-detection-job
                --region region \
                --job-id job ID
I
```

You will get the following JSON in response:

```
{
    "TopicsDetectionJobProperties": {
        "InputDataConfig": {
            "S3Uri": "s3://input bucket/input path",
            "InputFormat": "ONE_DOC_PER_LINE"
        },
        "NumberOfTopics": topics to return,
        "JobId": "job ID",
        "JobStatus": "COMPLETED",
        "JobName": "job name",
        "SubmitTime": timestamp,
```

```
        "OutputDataConfig": {
            "S3Uri": "s3://output bucket/ouput path"
        },
        "EndTime": timestamp
    }
}
```

## Topic Modeling Using the AWS SDK for Java

The following Java program detects the topics in a document collection. It uses the StartTopicsDetectionJob (p. 191) operation to start detecting topics. Next, it uses the DescribeTopicsDetectionJob (p. 134) operation to check the status of the topic detection. Finally, it calls ListTopicsDetectionJobs (p. 171) to show a list of all jobs submitted for the account.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.DescribeTopicsDetectionJobRequest;
import com.amazonaws.services.comprehend.model.DescribeTopicsDetectionJobResult;
import com.amazonaws.services.comprehend.model.InputDataConfig;
import com.amazonaws.services.comprehend.model.InputFormat;
import com.amazonaws.services.comprehend.model.ListTopicsDetectionJobsRequest;
import com.amazonaws.services.comprehend.model.ListTopicsDetectionJobsResult;
import com.amazonaws.services.comprehend.model.StartTopicsDetectionJobRequest;
import com.amazonaws.services.comprehend.model.StartTopicsDetectionJobResult;

public class App
{
    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();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        final String inputS3Uri = "s3://input bucket/input path";
        final InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
        final String outputS3Uri = "s3://output bucket/output path";
        final String dataAccessRoleArn = "arn:aws:iam::account ID:role/data access role";
        final int numberOfTopics = 10;

        final StartTopicsDetectionJobRequest startTopicsDetectionJobRequest = new
 StartTopicsDetectionJobRequest()
                .withInputDataConfig(new InputDataConfig()
                        .withS3Uri(inputS3Uri)
                        .withInputFormat(inputDocFormat))
                .withOutputDataConfig(new OutputDataConfig()
                        .withS3Uri(outputS3Uri))
                .withDataAccessRoleArn(dataAccessRoleArn)
                .withNumberOfTopics(numberOfTopics);

        final StartTopicsDetectionJobResult startTopicsDetectionJobResult =
 comprehendClient.startTopicsDetectionJob(startTopicsDetectionJobRequest);

        final String jobId = startTopicsDetectionJobResult.getJobId();
```

```
        System.out.println("JobId: " + jobId);

        final DescribeTopicsDetectionJobRequest describeTopicsDetectionJobRequest = new
 DescribeTopicsDetectionJobRequest()
                .withJobId(jobId);

        final DescribeTopicsDetectionJobResult describeTopicsDetectionJobResult =
 comprehendClient.describeTopicsDetectionJob(describeTopicsDetectionJobRequest);
        System.out.println("describeTopicsDetectionJobResult: " +
 describeTopicsDetectionJobResult);

        ListTopicsDetectionJobsResult listTopicsDetectionJobsResult =
 comprehendClient.listTopicsDetectionJobs(new ListTopicsDetectionJobsRequest());
        System.out.println("listTopicsDetectionJobsResult: " +
 listTopicsDetectionJobsResult);

    }
}
```

## Topic Modeling Using the AWS SDK for Python (Boto)

The following Python program detects the topics in a document collection. It uses the StartTopicsDetectionJob (p. 191) operation to start detecting topics. Next, it uses the DescribeTopicsDetectionJob (p. 134) operation to check the status of the topic detection. Finally, it calls ListTopicsDetectionJobs (p. 171) to show a list of all jobs submitted for the account.

```
import boto3
import json
from bson import json_util

comprehend = boto3.client(service_name='comprehend', region_name='region')

input_s3_url = "s3://input bucket/input path"
input_doc_format = "ONE_DOC_PER_FILE"
output_s3_url = "s3://output bucket/output path"
data_access_role_arn = "arn:aws:iam::account ID:role/data access role"
number_of_topics = 10

input_data_config = {"S3Uri": input_s3_url, "InputFormat": input_doc_format}
output_data_config = {"S3Uri": output_s3_url}

start_topics_detection_job_result =
 comprehend.start_topics_detection_job(NumberOfTopics=number_of_topics,

 InputDataConfig=input_data_config,

 OutputDataConfig=output_data_config,

 DataAccessRoleArn=data_access_role_arn)

print('start_topics_detection_job_result: ' +
 json.dumps(start_topics_detection_job_result))

job_id = start_topics_detection_job_result["JobId"]

print('job_id: ' + job_id)

describe_topics_detection_job_result =
 comprehend.describe_topics_detection_job(JobId=job_id)

print('describe_topics_detection_job_result: ' +
 json.dumps(describe_topics_detection_job_result, default=json_util.default))
```

```
list_topics_detection_jobs_result = comprehend.list_topics_detection_jobs()

print('list_topics_detection_jobs_result: ' + json.dumps(list_topics_detection_jobs_result,
 default=json_util.default))
```

## Topic Modeling Using the AWS SDK for .NET

The following C# program detects the topics in a document collection. It uses the StartTopicsDetectionJob (p. 191) operation to start detecting topics. Next, it uses the DescribeTopicsDetectionJob (p. 134) operation to check the status of the topic detection. Finally, it calls ListTopicsDetectionJobs (p. 171) to show a list of all jobs submitted for the account.

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        // Helper method for printing properties
        static private void PrintJobProperties(TopicsDetectionJobProperties props)
        {
            Console.WriteLine("JobId: {0}, JobName: {1}, JobStatus: {2}, NumberOfTopics:
 {3}\nInputS3Uri: {4}, InputFormat: {5}, OutputS3Uri: {6}",
                props.JobId, props.JobName, props.JobStatus, props.NumberOfTopics,
                props.InputDataConfig.S3Uri, props.InputDataConfig.InputFormat,
 props.OutputDataConfig.S3Uri);
        }

        static void Main(string[] args)
        {
            String text = "It is raining today in Seattle";

            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            String inputS3Uri = "s3://input bucket/input path";
            InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
            String outputS3Uri = "s3://output bucket/output path";
            String dataAccessRoleArn = "arn:aws:iam::account ID:role/data access role";
            int numberOfTopics = 10;

            StartTopicsDetectionJobRequest startTopicsDetectionJobRequest = new
 StartTopicsDetectionJobRequest()
            {
                InputDataConfig = new InputDataConfig()
                {
                    S3Uri = inputS3Uri,
                    InputFormat = inputDocFormat
                },
                OutputDataConfig = new OutputDataConfig()
                {
                    S3Uri = outputS3Uri
                },
                DataAccessRoleArn = dataAccessRoleArn,
                NumberOfTopics = numberOfTopics
            };
```

```
            StartTopicsDetectionJobResponse startTopicsDetectionJobResponse =
 comprehendClient.StartTopicsDetectionJob(startTopicsDetectionJobRequest);

            String jobId = startTopicsDetectionJobResponse.JobId;
            Console.WriteLine("JobId: " + jobId);

            DescribeTopicsDetectionJobRequest describeTopicsDetectionJobRequest = new
 DescribeTopicsDetectionJobRequest()
            {
                JobId = jobId
            };

            DescribeTopicsDetectionJobResponse describeTopicsDetectionJobResponse =
 comprehendClient.DescribeTopicsDetectionJob(describeTopicsDetectionJobRequest);

 PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

            ListTopicsDetectionJobsResponse listTopicsDetectionJobsResponse =
 comprehendClient.ListTopicsDetectionJobs(new ListTopicsDetectionJobsRequest());
            foreach (TopicsDetectionJobProperties props in
 listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
                PrintJobProperties(props);
        }
    }
}
```

# Using the Batch APIs

To send batches of up to 25 documents, you can use the Amazon Comprehend batch operations. Calling a batch operation is identical to calling the single document APIs for each document in the request. Using the batch APIs can result in better performance for your applications. For more information, see Multiple Document Synchronous Processing (p. 63).

**Topics**



## Batch Processing With the SDK for Java

The following sample program shows how to use the BatchDetectEntities (p. 97) operation with the SDK for Java. The response from the server contains a BatchDetectEntitiesItemResult (p. 204) object for each document that was successfully processed. If there is an error processing a document there will be a record in the error list in the response. The example gets each of the documents with an error and resends them.

```
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.comprehend.AmazonComprehend;
import com.amazonaws.services.comprehend.AmazonComprehendClientBuilder;
import com.amazonaws.services.comprehend.model.BatchDetectEntitiesItemResult;
import com.amazonaws.services.comprehend.model.BatchDetectEntitiesRequest;
import com.amazonaws.services.comprehend.model.BatchDetectEntitiesResult;
import com.amazonaws.services.comprehend.model.BatchItemError;

public class App
{
    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();

        AmazonComprehend comprehendClient =
            AmazonComprehendClientBuilder.standard()
                                         .withCredentials(awsCreds)
                                         .withRegion("region")
                                         .build();

        String[] textList = {"I love Seattle", "Today is Sunday", "Tomorrow is Monday", "I
love Seattle"};

        // Call detectEntities API
        System.out.println("Calling BatchDetectEntities");
        BatchDetectEntitiesRequest batchDetectEntitiesRequest = new
BatchDetectEntitiesRequest().withTextList(textList)

    .withLanguageCode("en");
        BatchDetectEntitiesResult batchDetectEntitiesResult  =
client.batchDetectEntities(batchDetectEntitiesRequest);

        for(BatchDetectEntitiesItemResult item : batchDetectEntitiesResult.getResultList())
{
            System.out.println(item);
        }

        // check if we need to retry failed requests
        if (batchDetectEntitiesResult.getErrorList().size() != 0)
        {
            System.out.println("Retrying Failed Requests");
            ArrayList<String> textToRetry = new ArrayList<String>();
            for(BatchItemError errorItem : batchDetectEntitiesResult.getErrorList())
            {
                textToRetry.add(textList[errorItem.getIndex()]);
            }

            batchDetectEntitiesRequest = new
BatchDetectEntitiesRequest().withTextList(textToRetry).withLanguageCode("en");
            batchDetectEntitiesResult  =
client.batchDetectEntities(batchDetectEntitiesRequest);

            for(BatchDetectEntitiesItemResult item :
batchDetectEntitiesResult.getResultList()) {
                System.out.println(item);
            }

        }
        System.out.println("End of DetectEntities");
    }
}
```

## Batch Processing With the AWS SDK for .NET

The following sample program shows how to use the BatchDetectEntities (p. 97) operation with the AWS SDK for .NET. The response from the server contains a BatchDetectEntitiesItemResult (p. 204) object for each document that was successfully processed. If there is an error processing a document there will be a record in the error list in the response. The example gets each of the documents with an error and resends them.

The .NET example in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for

deploying applications and managing services. For a .NET developer perspective of AWS, see the AWS Guide for .NET Developers.

```
using System;
using System.Collections.Generic;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        // Helper method for printing properties
        static private void PrintEntity(Entity entity)
        {
            Console.WriteLine("     Text: {0}, Type: {1}, Score: {2}, BeginOffset: {3}
 EndOffset: {4}",
                entity.Text, entity.Type, entity.Score, entity.BeginOffset,
 entity.EndOffset);
        }

        static void Main(string[] args)
        {
            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            List<String> textList = new List<String>()
            {
                { "I love Seattle" },
                { "Today is Sunday" },
                { "Tomorrow is Monday" },
                { "I love Seattle" }
            };

            // Call detectEntities API
            Console.WriteLine("Calling BatchDetectEntities");
            BatchDetectEntitiesRequest batchDetectEntitiesRequest = new
 BatchDetectEntitiesRequest()
            {
                TextList = textList,
                LanguageCode = "en"
            };
            BatchDetectEntitiesResponse batchDetectEntitiesResponse =
 comprehendClient.BatchDetectEntities(batchDetectEntitiesRequest);

            foreach (BatchDetectEntitiesItemResult item in
 batchDetectEntitiesResponse.ResultList)
            {
                Console.WriteLine("Entities in {0}:", textList[item.Index]);
                foreach (Entity entity in item.Entities)
                    PrintEntity(entity);
            }

            // check if we need to retry failed requests
            if (batchDetectEntitiesResponse.ErrorList.Count != 0)
            {
                Console.WriteLine("Retrying Failed Requests");
                List<String> textToRetry = new List<String>();
                foreach(BatchItemError errorItem in batchDetectEntitiesResponse.ErrorList)
                    textToRetry.Add(textList[errorItem.Index]);

                batchDetectEntitiesRequest = new BatchDetectEntitiesRequest()
                {
                    TextList = textToRetry,
                    LanguageCode = "en"
```

```
                };

                batchDetectEntitiesResponse =
 comprehendClient.BatchDetectEntities(batchDetectEntitiesRequest);

                foreach(BatchDetectEntitiesItemResult item in
 batchDetectEntitiesResponse.ResultList)
                {
                    Console.WriteLine("Entities in {0}:", textList[item.Index]);
                    foreach (Entity entity in item.Entities)
                        PrintEntity(entity);
                }
            }
            Console.WriteLine("End of DetectEntities");
        }
    }
}
```

# Batch Processing With the AWS CLI

These examples show how to use the batch API operations using the AWS Command Line Interface. All of the operations except `BatchDetectDominantLanguage` use the following JSON file called `process.json` as input. For that operation the `LanguageCode` entity is not included.

The third document in the JSON file (`"$$$$$$$$$"`) will cause an error during batch processing. It is included so that the operations will include a BatchItemError (p. 208) in the response.

```
{
   "LanguageCode": "en",
   "TextList": [
      "I have been living in Seattle for almost 4 years",
      "It is raining today in Seattle",
      "$$$$$$$$$"
   ]
}
```

The examples are formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

**Topics**

- Detect the Dominant Language Using a Batch (AWS CLI) (p. 45)
- Detect Entities Using a Batch (AWS CLI) (p. 46)
- Detect Key Phrases Using a Batch (AWS CLI) (p. 46)
- Detect Sentiment Using a Batch (AWS CLI) (p. 46)

## Detect the Dominant Language Using a Batch (AWS CLI)

The BatchDetectDominantLanguage (p. 94) operation determines the dominant language of each document in a batch. For a list of the languages that Amazon Comprehend can detect, see Detect the Dominant Language (p. 48). The following AWS CLI command calls the `BatchDetectDominantLanguage` operation.

```
aws comprehend batch-detect-dominant-language \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

The following is the response from the `BatchDetectDominantLanguage` operation:

```
{
    "ResultList": [
        {
          "Index": 0,
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.99
            }
          ]
        },
        {
          "Index": 1
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.82
            }
          ]
        }
    ],
    "ErrorList": [
      {
        "Index": 2,
        "ErrorCode": "InternalServerException",
        "ErrorMessage": "Unexpected Server Error. Please try again."
      }
    ]
}
```

## Detect Entities Using a Batch (AWS CLI)

Use the BatchDetectEntities (p. 97) operation to find the entities present in a batch of documents. For more information about entities, see Detect Entities (p. 50). The following AWS CLI command calls the `BatchDetectEntities` operation.

```
aws comprehend batch-detect-entities \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

## Detect Key Phrases Using a Batch (AWS CLI)

The BatchDetectKeyPhrases (p. 100) operation returns the key noun phrases in a batch of documents. The following AWS CLI command calls the `BatchDetectKeyNounPhrases` operation.

```
aws comprehend batch-detect-key-phrases
    --endpoint endpoint
    --region region
    --cli-input-json file://path to input file/process.json
```

## Detect Sentiment Using a Batch (AWS CLI)

Detect the overall sentiment of a batch of documents using the BatchDetectSentiment (p. 103) operation. The following AWS CLI command calls the `BatchDetectSentiment` operation.

```
aws comprehend batch-detect-sentiment \
    --endpoint endpoint \
    --region region \
```

```
    --cli-input-json file://path to input file/process.json
```

# Text Analysis APIs

Amazon Comprehend enables you to examine your documents to gain various insights about their content using a number of pre-trained models.

With Amazon Comprehend, you can perform the following on your documents:

- Detect the Dominant Language (p. 48)—Examine text to determine the dominant language.
- Detect Entities (p. 50)—Detect textual references to the names of people, places, and items as well as references to dates and quantities.
- Locate Key Phrases (p. 51)—Find key phrases such as "good morning" in a document or set of documents.
- Determine the Sentiment (p. 52)—Analyze documents and determine the dominant sentiment of the text.
- Analyze Syntax (p. 53)—Parse the words in your text and show the speech syntax for each word and enable you to understand the content of the document.
- Topic Modeling (p. 55)—Search the content of documents to determine common themes and topics.

Documents can be processed through these features in several ways: singly or in groups of up to 25 documents run synchronously with the result returned immediately, or in a larger batch run asynchronously, with the result saved to an S3 bucket. For more information, see Document Processing Modes (p. 59).

## Detect the Dominant Language

You can use Amazon Comprehend to examine text to determine the dominant language. Amazon Comprehend identifies the language using identifiers from RFC 5646 — if there is a 2-letter ISO 639-1 identifier, with a regional subtag if necessary, it uses that. Otherwise, it uses the ISO 639-2 3-letter code. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

The response includes a score that indicates the confidence level that Amazon Comprehend has that a particular language is the dominant language in the document. Each score is independent of the other scores — it does not indicate that a language makes up a particular percentage of a document.

If a long document, like a book, is written in multiple languages, you can break the long document into smaller pieces and run the `DetectDominantLanguage` operation on the individual pieces. You can then aggregate the results to determine the percentage of each language in the longer document.

Amazon Comprehend can detect the following languages.

| Code | Language | Code | Language | Code | Language |
|---|---|---|---|---|---|
| af | Afrikaans | hy | Armenian | ps | Pushto |
| am | Amharic | ilo | Iloko | qu | Quechua |
| ar | Arabic | id | Indonesian | ro | Romanian |
| as | Assamese | is | Icelandic | ru | Russian |
| az | Azerbaijani | it | Italian | sa | Sanskrit |
| ba | Bashkir | jv | Javanese | si | Sinhala |

| Code | Language | Code | Language | Code | Language |
|---|---|---|---|---|---|
| be | Belarusian | ja | Japanese | sk | Slovak |
| bn | Bengali | kn | Kannada | sl | Slovenian |
| bs | Bosnian | ka | Georgian | sd | Sindhi |
| bg | Bulgarian | kk | Kazakh | so | Somali |
| ca | Catalan | km | Central Khmer | es | Spanish |
| ceb | Cebuano | ky | Kirghiz | sq | Albanian |
| cs | Czech | ko | Korean | sr | Serbian |
| cv | Chuvash | ku | Kurdish | su | Sundanese |
| cy | Welsh | la | Latin | sw | Swahili |
| da | Danish | lv | Latvian | sv | Swedish |
| de | German | lt | Lithuanian | ta | Tamil |
| el | Greek | lb | Luxembourgish | tt | Tatar |
| en | English | ml | Malayalam | te | Telugu |
| eo | Esperanto | mr | Marathi | tg | Tajik |
| et | Estonian | mk | Macedonian | tl | Tagalog |
| eu | Basque | mg | Malagasy | th | Thai |
| fa | Persian | mn | Mongolian | tk | Turkmen |
| fi | Finnish | ms | Malay | tr | Turkish |
| fr | French | my | Burmese | ug | Uighur |
| gd | Scottish Gaelic | ne | Nepali | uk | Ukrainian |
| ga | Irish | new | Newari | ur | Urdu |
| gl | Galician | nl | Dutch | uz | Uzbek |
| gu | Gujarati | no | Norwegian | vi | Vietnamese |
| ht | Haitian | or | Oriya | yi | Yiddish |
| he | Hebrew | pa | Punjabi | yo | Yoruba |
| hi | Hindi | pl | Polish | zh | Chinese (Simplified) |
| hr | Croatian | pt | Portuguese | zh-TW | Chinese (Traditional) |
| hu | Hungarian | | | | |

You can use any of the following operations to detect the dominant language in a document or set of documents.

- DetectDominantLanguage (p. 136)

- BatchDetectDominantLanguage (p. 94)
- StartDominantLanguageDetectionJob (p. 178)

The `DetectDominantLanguage` operation returns a DominantLanguage (p. 218) object. The `BatchDetectDominantLanguage` operation returns a list of `DominantLanguage` objects, one for each document in the batch. The `StartDominantLanguageDetectionJob` operation starts an asynchronous job that produces a file containing a list of `DominantLanguage` objects, one for each document in the job.

The following example is the response from the `DetectDominantLanguage` operation.

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9793661236763
        }
    ]
}
```

# Detect Entities

Use the DetectEntities (p. 139), BatchDetectEntities (p. 97), and StartEntitiesDetectionJob (p. 181) operations to detect entities in a document. An *entity* is a textual reference to the unique name of a real-world object such as people, places, and commercial items, and to precise references to measures such as dates and quantities.

For example, in the text "John moved to 1313 Mockingbird Lane in 2012," "John" might be recognized as a `PERSON`, "1313 Mockingbird Lane" might be recognized as a `LOCATION`, and "2012" might be recognized as a `DATE`.

Each entity also has a score that indicates the level of confidence that Amazon Comprehend has that it correctly detected the entity type. You can filter out the entities with lower scores to reduce the risk of using incorrect detections.

The following table lists the entity types.

| Type | Description |
|---|---|
| COMMERCIAL_ITEM | A branded product |
| DATE | A full date (for example, 11/25/2017), day (Tuesday), month (May), or time (8:30 a.m.) |
| EVENT | An event, such as a festival, concert, election, etc. |
| LOCATION | A specific location, such as a country, city, lake, building, etc. |
| ORGANIZATION | Large organizations, such as a government, company, religion, sports team, etc. |
| OTHER | Entities that don't fit into any of the other entity categories |
| PERSON | Individuals, groups of people, nicknames, fictional characters |

| Type | Description |
| --- | --- |
| QUANTITY | A quantified amount, such as currency, percentages, numbers, bytes, etc. |
| TITLE | An official name given to any creation or creative work, such as movies, books, songs, etc. |

You can use any of the following operations to detect entities in a document or set of documents.

- DetectEntities (p. 139)
- BatchDetectEntities (p. 97)
- StartEntitiesDetectionJob (p. 181)

The operations return a list of Entity (p. 225) objects, one for each entity in the document. The `BatchDetectEntities` operation returns a list of `Entity` objects, one list for each document in the batch. The `StartEntitiesDetectionJob` operation starts an asynchronous job that produces a file containing a list of `Entity` objects for each document in the job.

The following example is the response from the `DetectEntities` operation.

```
{
    "Entities": [
        {
            "Text": "today",
            "Score": 0.97,
            "Type": "DATE",
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.95,
            "Type": "LOCATION",
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ],
    "LanguageCode": "en"
}
```

# Locate Key Phrases

You can use Amazon Comprehend operations to find key phrases in your document.

A *key phrase* is a string containing a noun phrase that describes a particular thing. It generally consists of a noun and the modifiers that distinguish it. For example, "day" is a noun; "a beautiful day" is a noun phrase that includes an article ("a") and an adjective ("beautiful"). Each key phrase includes a score that indicates the level of confidence that Amazon Comprehend has that the string is a noun phrase. You can use the score to determine if the detection has high enough confidence for your application.

You can use any of the following operations to detect key phrases in a document or set of documents.

- DetectKeyPhrases (p. 142)
- BatchDetectKeyPhrases (p. 100)
- StartKeyPhrasesDetectionJob (p. 185)

The operations return a list of KeyPhrase (p. 239) objects, one for each key phrase in the document. The `BatchDetectKeyPhrases` operation returns a list of `KeyPhrase` objects, one for each document in the batch. The `StartKeyPhrasesDetectionJob` operation starts an asynchronous job that produces a file containing a list of `KeyPhrase` objects for each document in the job.

The following example is the response from the `DetectKeyPhrases` operation.

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ]
}
```

# Determine the Sentiment

Use Amazon Comprehend to determine the sentiment of a document. You can determine if the sentiment is positive, negative, neutral, or mixed. For example, you can use sentiment analysis to determine the sentiments of comments on a blog posting to determine if your readers liked the post.

You can use any of the following operations to detect the sentiment of a document or a set of documents.

- DetectSentiment (p. 145)
- BatchDetectSentiment (p. 103)
- StartSentimentDetectionJob (p. 188)

The operations return the most likely sentiment for the text as well as the scores for each of the sentiments. The score represents the likelihood that the sentiment was correctly detected. For example, in the example below it is 95 percent likely that the text has a `Positive` sentiment. There is a less than 1 percent likelihood that the text has a `Negative` sentiment. You can use the `SentimentScore` to determine if the accuracy of the detection meets the needs of your application.

The `DetectSentiment` operation returns an object that contains the detected sentiment and a SentimentScore (p. 248) object. The `BatchDetectSentiment` operation returns a list of sentiments and `SentimentScore` objects, one for each document in the batch. The `StartSentimentDetectionJob` operation starts an asynchronous job that produces a file containing a list of sentiments and `SentimentScore` objects, one for each document in the job.

The following example is the response from the `DetectSentiment` operation.

```
{
"SentimentScore": {
        "Mixed": 0.030585512690246105,
        "Positive": 0.94992071056365967,
        "Neutral": 0.0141543131828308,
```

```
        "Negative": 0.00893945890665054
    },
    "Sentiment": "POSITIVE",
    "LanguageCode": "en"
}
```

# Analyze Syntax

Use the DetectSyntax (p. 148) and BatchDetectSyntax (p. 106) operations to analyze your documents to parse the words from the document and return the part of speech, or syntactic function, for each word in the document. You can identify the nouns, verbs, adjectives and so on in your document. Use this information to gain a richer understanding of the content of your documents, and to understand the relationship of the words in the document.

For example, you can look for the nouns in a document and then look for the verbs related to those nouns. In a sentence like "My grandmother moved her couch" you can see the nouns, "grandmother" and "couch," and the verb, "moved." You can use this information to build applications for analyzing text for word combinations that you are interested in.

To start the analysis, Amazon Comprehend parses the source text to find the individual words in the text. After the text is parsed, each word is assigned the part of speech that it takes in the source text.

Amazon Comprehend can identify 17 parts of speech. The parts of speech recognized are:

| Token | Part of speech |
|---|---|
| ADJ | Adjective<br><br>Words that typically modify nouns. |
| ADP | Adposition<br><br>The head of a prepositional or postpositional phrase. |
| ADV | Adverb<br><br>Words that typically modify verbs. They may also modify adjectives and other adverbs. |
| AUX | Auxiliary<br><br>Function words that accompanies the verb of a verb phrase. |
| CCONJ | Coordinating conjunction<br><br>Words that links words or phrases without subordinating one to the other. |
| DET | Determiner<br><br>Articles and other words that specify a particular noun phrase. |

| Token | Part of speech |
| --- | --- |
| INTJ | Interjection<br><br>Words used as an exclamation or part of an exclamation. |
| NOUN | Noun<br><br>Words that specify a person, place, thing, animal, or idea. |
| NUM | Numeral<br><br>Words, typically determiners, adjectives, or pronouns, that express a number. |
| O | Other<br><br>Words that can't be assigned a part of speech category. |
| PART | Particle<br><br>Function words associated with another word or phrase to impart meaning. |
| PRON | Pronoun<br><br>Words that substitute for nouns or noun phrases. |
| PROPN | Proper noun<br><br>A noun that is the name of a specific individual, place or object. |
| PUNCT | Punctuation<br><br>Non-alphabetical characters that delimit text. |
| SCONJ | Subordinating conjunction<br><br>A conjunction that links parts of sentences by make one of them part of the other. |
| SYM | Symbol<br><br>Word-like entities such as the dollar sign ($) or mathematical symbols. |

| Token | Part of speech |
| --- | --- |
| VERB | Verb<br><br>Words that signal events and actions. |

For more information about the parts of speech, see Universal POS tags at the *Universal Dependencies* website.

The operations return tokens that identify the word and the part of speech that the word represents in the text. Each token represents a word in the source text. It provides the location of the word in the source, the part of speech that the word takes in the text, the confidence that Amazon Comprehend has that the part of speech was correctly identified, and the word that was parsed from the source text.

The following is the structure of the list of syntax tokens. One syntax token is generated for each word in the document.

```
{
   "SyntaxTokens": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "PartOfSpeech": {
            "Score": number,
            "Tag": "string"
         },
         "Text": "string",
         "TokenId": number
      }
   ]
}
```

Each token provides the following information:

- `BeginOffset` and `EndOffset`—Provides the location of the word in the input text.
- `PartOfSpeech`—Provides two pieces of information, the `Tag` that identifies the part of speech and the `Score` that represents the confidence that Amazon Comprehend Syntax has that the part of speech was correctly identifies.
- `Text`—Provides the word that was identified.
- `TokenId`—Provides an identifier for the token. The identifier is the position of the token in the list of tokens.

# Topic Modeling

You can use Amazon Comprehend to examine the content of a collection of documents to determine common themes. For example, you can give Amazon Comprehend a collection of news articles, and it will determine the subjects, such as sports, politics, or entertainment. The text in the documents doesn't need to be annotated.

Amazon Comprehend uses a Latent Dirichlet Allocation-based learning model to determine the topics in a set of documents. It examines each document to determine the context and meaning of a word. The set of words that frequently belong to the same context across the entire document set make up a topic.

A word is associated to a topic in a document based on how prevalent that topic is in a document and how much affinity the topic has to the word. The same word can be associated with different topics in different documents based on the topic distribution in a particular document.

For example, the word "glucose" in an article that talks predominantly about sports can be assigned to the topic "sports," while the same word in an article about "medicine" will be assigned to the topic "medicine."

Each word associated with a topic is given a weight that indicates how much the word helps define the topic. The weight is an indication of how many times the word occurs in the topic compared to other words in the topic, across the entire document set.

For the most accurate results you should provide Amazon Comprehend with the largest possible corpus to work with. For best results:

- You should use at least 1,000 documents in each topic modeling job.
- Each document should be at least 3 sentences long.
- If a document consists of mostly numeric data, you should remove it from the corpus.

Topic modeling is an asynchronous process. You submit your list of documents to Amazon Comprehend from an Amazon S3 bucket using the StartTopicsDetectionJob (p. 191) operation. The response is sent to an Amazon S3 bucket. You can configure both the input and output buckets. Get a list of the topic modeling jobs that you have submitted using the ListTopicsDetectionJobs (p. 171) operation and view information about a job using the DescribeTopicsDetectionJob (p. 134) operation. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Documents must be in UTF-8 formatted text files. You can submit your documents two ways. The following table shows the options.

| Format | Description |
| --- | --- |
| One document per file | Each file contains one input document. This is best for collections of large documents. |
| One document per line | The input is a single file. Each line in the file is considered a document. This is best for short documents, such as social media postings.<br><br>Each line must end with a line feed (LF, \n), a carriage return (CR, \r), or both (CRLF, \r\n). The Unicode line separator (u+2028) can't be used to end a line. |

For more information, see the InputDataConfig (p. 238) data type.

After Amazon Comprehend processes your document collection, it returns a compressed archive containing two files, `topic-terms.csv` and `doc-topics.csv`. For more information about the output file, see OutputDataConfig (p. 243).

The first output file, `topic-terms.csv`, is a list of topics in the collection. For each topic, the list includes, by default, the top terms by topic according to their weight. For example, if you give Amazon Comprehend a collection of newspaper articles, it might return the following to describe the first two topics in the collection:

| Topic | Term | Weight |
|---|---|---|
| 000 | team | 0.118533 |
| 000 | game | 0.106072 |
| 000 | player | 0.031625 |
| 000 | season | 0.023633 |
| 000 | play | 0.021118 |
| 000 | yard | 0.024454 |
| 000 | coach | 0.016012 |
| 000 | games | 0.016191 |
| 000 | football | 0.015049 |
| 000 | quarterback | 0.014239 |
| 001 | cup | 0.205236 |
| 001 | food | 0.040686 |
| 001 | minutes | 0.036062 |
| 001 | add | 0.029697 |
| 001 | tablespoon | 0.028789 |
| 001 | oil | 0.021254 |
| 001 | pepper | 0.022205 |
| 001 | teaspoon | 0.020040 |
| 001 | wine | 0.016588 |
| 001 | sugar | 0.015101 |

The weights represent a probability distribution over the words in a given topic. Since Amazon Comprehend returns only the top 10 words for each topic the weights won't sum to 1.0. In the rare cases where there are less than 10 words in a topic, the weights will sum to 1.0.

The words are sorted by their discriminative power by looking at their occurrence across all topics. Typically this is the same as their weight, but in some cases, such as the words "play" and "yard" in the table, this results in an order that is not the same as the weight.

You can specify the number of topics to return. For example, if you ask Amazon Comprehend to return 25 topics, it returns the 25 most prominent topics in the collection. Amazon Comprehend can detect up to 100 topics in a collection. Choose the number of topics based on your knowledge of the domain. It may take some experimentation to arrive at the correct number.

The second file, `doc-topics.csv`, lists the documents associated with a topic and the proportion of the document that is concerned with the topic. If you specified `ONE_DOC_PER_FILE` the document is identified by the file name. If you specified `ONE_DOC_PER_LINE` the document is identified by the file name and the 0-indexed line number within the file. For example, Amazon Comprehend might return the following for a collection of documents submitted with one document per file:

| Document | Topic | Proportion |
|---|---|---|
| sample-doc1 | 000 | 0.999330137 |
| sample-doc2 | 000 | 0.998532187 |
| sample-doc3 | 000 | 0.998384574 |
| ... | | |
| sample-docN | 000 | 3.57E-04 |

Amazon Comprehend utilizes information from the *Lemmatization Lists Dataset by MBM*, which is made available here under the Open Database License (ODbL) v1.0.

# Document Processing Modes

Amazon Comprehend enables you to examine your documents to gain various insights about their content.

When evaluating your documents, you can use one of several methods to process them, depending on how many documents you have and how you want to view the results:

- Single-Document Processing (p. 59)—You call Amazon Comprehend with a single document and receive a synchronous response, delivered to your application right away.
- Multiple Document Synchronous Processing (p. 63)—You call Amazon Comprehend with a collection of up to 25 documents and receive a synchronous response.
- Asynchronous Batch Processing (p. 59)—You put a collection of documents into an Amazon S3 bucket and start an asynchronous operation to analyze the documents. The results of the analysis are returned in an S3 bucket.

## Single-Document Processing

The single-document operations are synchronous operations that return the results of analyzing the document directly to your application. You should use the single-document operations when you are creating an interactive application that works on one document at a time.

You can use the following single-document operations:

- DetectDominantLanguage (p. 136)
- DetectEntities (p. 139)
- DetectKeyPhrases (p. 142)
- DetectSentiment (p. 145)
- DetectSyntax (p. 148)

## Asynchronous Batch Processing

To analyze large documents and large collections of documents, use the Amazon Comprehend asynchronous operations. There is an asynchronous version of each of the Amazon Comprehend operations and an additional set of operations for topic modeling.

To analyze a collection of documents, you typically perform the following steps:

1. Store the documents in an Amazon S3 bucket.
2. Start one or more jobs to analyze the documents.
3. Monitor the progress of an analysis job.
4. Retrieve the results of the analysis from an S3 bucket when the job is complete.

The following sections describe using the Amazon Comprehend API to run asynchronous operations.

# Prerequisites

Documents must be in UTF-8-formatted text files. You can submit your documents in two formats. The format you use depends on the type of documents you want to analyze, as described in the following table.

| Description | Format |
| --- | --- |
| Each file contains one input document. This is best for collections of large documents. | One document per file |
| The input is one or more files. Each line in a file is considered a document. This is best for short documents, such as social media postings.<br><br>Each line must end with a line feed (LF, \n), a carriage return (CR, \r), or both (CRLF, \r\n). You can't use the UTF-8 line separator (u+2028) to end a line. | One document per line |

When you start an analysis job, you specify the S3 location for your input data. The URI must be in the same AWS Region as the API endpoint that you are calling. The URI can point to a single file or it can be the prefix for a collection of data files. For more information, see the InputDataConfig (p. 238) data type.

You must grant Amazon Comprehend access to the Amazon S3 bucket that contains your document collection and output files. For more information, see Role-Based Permissions Required for Asynchronous Operations (p. 85).

# Starting an Analysis Job

To submit an analysis job, use either the Amazon Comprehend console or the appropriate `Start*` operation:

- StartDominantLanguageDetectionJob (p. 178)—Start a job to detect the dominant language in each document in the collection. For more information about the dominant language in a document, see Detect the Dominant Language (p. 48).
- StartEntitiesDetectionJob (p. 181)—Start a job to detect entities in each document in the collection. For more information about entities, see Detect Entities (p. 50).
- StartKeyPhrasesDetectionJob (p. 185)—Start a job to detect key phrases in each document in the collection. For more information about key phrases, see Locate Key Phrases (p. 51).
- StartSentimentDetectionJob (p. 188)—Start a job to detect the emotional sentiment in each document in the collection. For more information about sentiments, see Determine the Sentiment (p. 52).
- StartTopicsDetectionJob (p. 191)—Start a job to detect the topics in a document collection. For more information about topic modeling, see Topic Modeling (p. 55).

# Monitoring Analysis Jobs

The `Start*` operation returns an ID that you can use to monitor the job's progress.

To monitor progress using the API, you use one of two operations, depending on whether you want to monitor the progress of an individual job or multiple jobs.

To monitor the progress of an individual analysis job, use the `Describe*` operations. You provide the job ID returned by the `Start*` operation. The response from the `Describe*` operation contains the `JobStatus` field with the job's status.

To monitor the progress of multiple analysis jobs, use the `List*` operations. `List*` operations return a list of jobs that you submitted to Amazon Comprehend. The response includes a `JobStatus` field for each job that tells you the status of the job.

If the status field is set to `COMPLETED` or `FAILED`, job processing has completed.

To get the status of individual jobs, use the `Describe*` operation for the analysis that you are performing.

- DescribeDominantLanguageDetectionJob (p. 123)
- DescribeEntitiesDetectionJob (p. 125)
- DescribeKeyPhrasesDetectionJob (p. 130)
- DescribeSentimentDetectionJob (p. 132)
- DescribeTopicsDetectionJob (p. 134)

To get the status of a multiple jobs, use the `List*` operation for the analysis that you are performing.

- ListDominantLanguageDetectionJobs (p. 156)
- ListEntitiesDetectionJobs (p. 159)
- ListKeyPhrasesDetectionJobs (p. 165)
- ListSentimentDetectionJobs (p. 168)
- ListTopicsDetectionJobs (p. 171)

To restrict the results to jobs that match certain criteria, use the `List*` operations' `Filter` parameter. You can filter on the job name, the job status, and the date and time that the job was submitted. For more information, see the `Filter` parameter for each of the `List*` operations in the Actions (p. 92) reference.

# Getting Analysis Results

After an analysis job has finished, use a `Describe*` operation to get the location of the results. If the job status is `COMPLETED`, the response includes an `OutputDataConfig` field that contains a field with the Amazon S3 location of the output file. The file, `output.tar.gz`, is a compressed archive that contains the results of the analysis.

If the status of a job is `FAILED`, the response contains a `Message` field that describes the reason that the analysis job didn't complete successfully.

To get the status of individual jobs, use the appropriate `Describe*` operation:

- DescribeDominantLanguageDetectionJob (p. 123)
- DescribeEntitiesDetectionJob (p. 125)
- DescribeKeyPhrasesDetectionJob (p. 130)
- DescribeSentimentDetectionJob (p. 132)
- DescribeTopicsDetectionJob (p. 134)

The results are returned in a single file, with one JSON structure for each document. Each response file also includes error messages for any job with the status field set to `FAILED`.

Each of the following sections shows examples of output for the two input formats.

## Getting Dominant Language Detection Results

The following is an example of an output file from an analysis that detected the dominant language. The format of the input is one document per line. For more information, see the DetectDominantLanguage (p. 136) operation.

```
{"File": "0_doc", "Languages": [{"LanguageCode": "en", "Score": 0.9514502286911011},
 {"LanguageCode": "de", "Score": 0.02374090999364853}, {"LanguageCode": "nl", "Score":
 0.003208699868991971}, "Line": 0}
{"File": "1_doc", "Languages": [{"LanguageCode": "en", "Score": 0.9822712540626526},
 {"LanguageCode": "de", "Score": 0.002621392020955682}, {"LanguageCode": "es", "Score":
 0.002386554144322872}], "Line": 1}
```

The following is an example of output from an analysis where the format of the input is one document per file:

```
{"File": "small_doc", "Languages": [{"LanguageCode": "en", "Score": 0.9728053212165833},
 {"LanguageCode": "de", "Score": 0.007670710328966379}, {"LanguageCode": "es", "Score":
 0.0028472368139773607}]}
{"File": "huge_doc", "Languages": [{"LanguageCode": "en", "Score": 0.984955906867981},
 {"LanguageCode": "de", "Score": 0.0026436643674969673}, {"LanguageCode": "fr", "Score":
 0.001420688116922974 6}]}
```

## Getting Entity Detection Results

The following is an example of an output file from an analysis that detected entities in documents. The format of the input is one document per line. For more information, see the DetectEntities (p. 139) operation. The output contains two error messages, one for a document that is too long and one for a document that isn't in UTF-8 format.

```
{"File": "50_docs", "Line": 0, "Entities": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.9763959646224976, "Text": "Cluj-NapocaCluj-Napoca", "Type": "LOCATION"}"]}
{"File": "50_docs", "Line": 1, "Entities": [{"BeginOffset": 11, "EndOffset": 15, "Score":
 0.9615424871444702, "Text": "Maat", "Type": "PERSON"}}]}

{"File": "50_docs", "Line": 2, "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage":
 "Document size exceeds maximum size limit 102400 bytes."}
{"File": "50_docs", "Line": 3, "ErrorCode": "UNSUPPORTED_ENCODING", "ErrorMessage":
 "Document is not in UTF-8 format and all subsequent lines are ignored."}
```

The following is an example of output from an analysis where the format of the input is one document per file. The output contains two error messages, one for a document that is too long and one for a document that isn't in UTF-8 format.

```
{"File": "non_utf8.txt", "ErrorCode": "UNSUPPORTED_ENCODING", "ErrorMessage": "Document is
 not in UTF-8 format and all subsequent line are ignored."}

{"File": "small_doc", "Entities": [{"BeginOffset": 0, "EndOffset": 4, "Score":
 0.645766019821167, "Text": "Maat", "Type": "PERSON"}]}

{"File": "huge_doc", "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage": "Document size
 exceeds size limit 102400 bytes."}
```

## Getting Key Phrase Detection Results

The following is an example of an output file from an analysis that detected key phrases in a document. The format of the input is one document per line. For more information, see the DetectKeyPhrases (p. 142) operation.

```
{"File": "50_docs", "KeyPhrases": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.8948641419410706, "Text": "Cluj-NapocaCluj-Napoca"}, {"BeginOffset": 45, "EndOffset":
 49, "Score": 0.9989854693412781, "Text": "Cluj"}], "Line": 0}
```

The following is an example of the output from an analysis where the format of the input is one document per file.

```
{"File": "1_doc", "KeyPhrases": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.8948641419410706, "Text": "Cluj-NapocaCluj-Napoca"}, {"BeginOffset": 45, "EndOffset":
 49, "Score": 0.9989854693412781, "Text": "Cluj"}]}
```

## Getting Sentiment Detection Results

The following is an example of an output file from an analysis that detected the sentiment expressed in a document. It includes an error message because one document is too long. The format of the input is one document per line. For more information, see the DetectSentiment (p. 145) operation.

```
{"File": "50_docs", "Line": 0, "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.002734508365392685, "Negative": 0.008935936726629734, "Neutral": 0.9841893315315247,
 "Positive": 0.004140198230743408}}
{"File": "50_docs", "Line": 1, "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage":
 "Document size is exceeded maximum size limit 5120 bytes."}
{"File": "50_docs", "Line": 2, "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.0023119584657251835, "Negative": 0.0029857370536774397, "Neutral": 0.9866572022438049,
 "Positive": 0.008045154623687267}}
```

The following is an example of the output from an analysis where the format of the input is one document per file.

```
{"File": "small_doc", "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.0023450672160834074, "Negative": 0.0009663937962614, "Neutral": 0.9795311689376831,
 "Positive": 0.017157377675175667}}
{"File": "huge_doc", "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage": "Document size
 is exceeds the limit of 5120 bytes."}
```

## Getting Topic Modeling Results

Topic modeling analysis returns two files in the `output.tar.gz` file. For more information, see Topic Modeling (p. 55).

# Multiple Document Synchronous Processing

When you have multiple documents that you want to process, you can use the `Batch*` operations to send more than one document to Amazon Comprehend at a time. You can send up to 25 documents in each request. Amazon Comprehend sends back a list of responses, one for each document in the request.

You can use the following operations to process multiple documents in a single request. Requests made with these operations are synchronous. Your application calls the operation and then waits for the response from the service.

- BatchDetectDominantLanguage (p. 94)
- BatchDetectEntities (p. 97)
- BatchDetectKeyPhrases (p. 100)
- BatchDetectSentiment (p. 103)
- BatchDetectSyntax (p. 106)

Using the `Batch*` operations is identical to calling the single document APIs for each of the documents in the request. Using these APIs can result in better performance for your applications.

The input to each of the APIs is a JSON structure containing the documents to process. For all operations except `BatchDetectDominantLanguage`, you must set the input language. You can set only one input language for each request. For example, the following is the input to the `BatchDetectEntities` operation. It contains two documents and is in English.

```
{
   "LanguageCode": "en",
   "TextList": [
      "I have been living in Seattle for almost 4 years",
      "It is raining today in Seattle"
   ]
}
```

The response from a `Batch*` operation contains two lists, the `ResultList` and the `ErrorList`. The `ResultList` contains one record for each document that was successfully processed. The result for each document in the request is identical to the result you would get if you ran a single document operation on the document. The results for each document are assigned an index based on the order of the documents in the input file. The response from the `BatchDetectEntities` operation is:

```
{
   "ResultList"  : [
      {
         "Index": 0,
         "Entities": [
            {
               "Text": "Seattle",
               "Score": 0.95,
               "Type": "LOCATION",
               "BeginOffset": 22,
               "EndOffset": 29
            },
            {
               "Text": "almost 4 years",
               "Score": 0.89,
               "Type": "QUANTITY",
               "BeginOffset": 34,
               "EndOffset": 48
            }
         ]
      },
      {
         "Index": 1,
         "Entities": [
            {
              "Text": "today",
              "Score": 0.87,
```

```
              "Type": "DATE",
              "BeginOffset": 14,
              "EndOffset": 19
            },
            {
               "Text": "Seattle",
               "Score": 0.96,
               "Type": "LOCATION",
               "BeginOffset": 23,
               "EndOffset": 30
            }
         ]
      }
   ],
   "ErrorList": []
}
```

When an error occurs in the request the response contains an `ErrorList` that identifies the documents that contained an error. The document is identified by its index in the input list. For example, the following input to the `BatchDetectLanguage` operation contains a document that cannot be processed:

```
{
   "TextList": [
     "hello friend",
     "$$$$$$",
     "hola amigo"
   ]
}
```

The response from Amazon Comprehend includes an error list that identifies the document that contained an error:

```
{
    "ResultList": [
        {
          "Index": 0,
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.99
            }
          ]
        },
        {
          "Index": 2
          "Languages":[
            {
              "LanguageCode":"es",
              "Score": 0.82
            }
          ]
        }
    ],
    "ErrorList": [
      {
        "Index": 1,
        "ErrorCode": "InternalServerException",
        "ErrorMessage": "Unexpected Server Error. Please try again."
      }
    ]
}
```

# Comprehend Custom

Customize Comprehend for your specific requirements without the skillset required to build machine learning-based NLP solutions. Using automatic machine learning, or AutoML, Comprehend Custom builds customized NLP models on your behalf, using data you already have. Training and calling custom comprehend models are both async (batch) operations.

Amazon Comprehend uses a proprietary, state-of-the-art sequence tagging deep neural network model that powers the same Amazon Comprehend detect entities service to train your custom entity recognizer models. In addition, we understand that acquiring training data could be costly. To help customers build a highly accurate model with limited amount of data, Amazon Comprehend uses a technique called *transfer learning* to train your custom models based on an sophisticated general-purpose entities recognition model that was pre-trained with a large amount of data we collected from multiple domains. Offline experiments showed that transfer learning significantly improved custom entity recognizer model accuracy especially when the amount of training data is small.

**Topics**



# Custom Classification

You can use Amazon Comprehend to build your own models for *custom classification*, assigning a document to a class or a category.

For example, you can categorize the content of support requests so that you can route the request to the proper support team. Or you can categorize emails received from customers to provide guidance on the requests that customers are making. You can combine Amazon Comprehend with Amazon Transcribe to convert speech to text and then to classify the requests coming from support phone calls.

Custom classification is a two step process. First you train a custom classifier to recognize the categories that are of interest to you. To train the classifier, you send Amazon Comprehend a group of labeled documents. After Amazon Comprehend builds the classifier, you send documents to be classified. The custom classifier examines each document and returns the label that best represents the content of the document.

You can have multiple custom classifiers in your account, each trained using different data. You choose the classifier to use when you submit a classification job.

## Training a Custom Classifier

To train a custom classifier you:

1. Label your training data with the custom categories that you want the classifier to recognize.
2. Put your training data in an Amazon S3 bucket. The bucket must have AWS Identity and Access Management (IAM) Amazon Comprehend read permissions that allow Amazon Comprehend access. For more information, see Role-Based Permissions Required for Asynchronous Operations (p. 85). You must also designate another S3 bucket for your output with the same requirements

3. Submit a training job using the CreateDocumentClassifier (p. 109) operation.

You can train a custom classifier using any of the languages that work with Amazon Comprehend: English, Spanish, German, Italian, French, or Portuguese. However, you can only train the classifier in one language. Classifiers do not support multiple languages.

After you ask Amazon Comprehend to create a custom classifier, you can monitor the progress of the request using the ??? (p. 121) operation. Once the `Status` field is `TRAINED` you can then use the classifier to classify documents.

Each custom classifier that you create can only be trained for a one category

## Creating Training Data

To train the custom classifier, you need to have the labels you want (such as "PRICING", "DEFECT", "PROFANITY" and so on), and examples of documents for each of those labels. Each custom classifier you create categorize for multiple labels, based on your dataset, up to a total of 1000 unique labels. However, its strongly recommended that you have a sufficient number of training documents to accommodate the multiple labels.

> **Note**
> It is important to remember that even though you can use multiple labels in a classifier, no hiearchy is determined by them when you use the classifier on unlabeled document.

To prepare the training dataset, you build a CSV file that contains a label and document per line:

```
label,Text of document 1
label,Text of document 2
```

In each line, the label is placed first, followed by the document.

Labels can be virtually any valid UTF-8 string, be we strongly suggest labels that are clear and don't overlap in meaning. They must be uppercase, can be multitoken, have whitespace, consist of multiple words connect by underscores or hyphens or may even contain a comma in it, as long as it is correctly escaped.

The document can follow the same pattern, although it is not necessary to be uppercase. Training documents must be ended with with \n or \r\n and be a valid UTF-8 in a CSV file. Additionally, each document should contain a good representative distribution of the types of documents that the classifier must categorize in general practice.

We recommend that you train the model with up to 1,000 training documents for each label. While a minimum of 50 training documents for each label is required, you will get significantly better accuracy with more documents. The total size of the training documents must be less than 5 Gb and you can provide up to 1000 unique classes. In the example below there are 5 total examples (label, document pairs). There are 5 labels, CAT, CAT, DOG, CAT, FISH. There are **3 unique classes**: CAT, DOG, FISH. Amazon Comprehend custom classification supports up to 1 million examples containing up to 1000 unique classes.

| **column 1** | **column 2** |
|---|---|
| CAT | document text 1 |
| CAT | document text 2 |
| DOG | document text 3 |
| CAT | document text 4 |

| column 1 | column 2 |
|---|---|
| FISH | document text 5 |

Amazon Comprehend Amazon Comprehend will use between 10 and 20 percent of the documents that you submit for training to test the custom classifier. This testing material is randomly selected after we ensure that there are no labels in the test set which we have never seen before. For example, if the data contained 1000 instances of the CAT label, 1000 instances of the DOG, and a single instance of the FISH label, then the test set would approximately be 100 CAT, 100 DOG. The FISH label would not be included in the test set, as there is only a single example available. please note that it's highly unlikely you will see a doc labelled as FISH during prediction/inference in a setting like this.

The result of the testing is returned in the `Metrics` fields returned by the the section called "DescribeDocumentClassifier" (p. 121) operation.

Once you've finished training your model, the training metrics can provide you with information that you can use to decide if the model is trained sufficiently for your needs.

Once the training job completes, the service provides an endpoint to a trained model that is now used to organize unlabeled documents.

# Running a Classification Job

Once you've trained your model, your custom classifier is available for use in categorizing unlabeled documents.

For Amazon Comprehend to process them, all documents must be in UTF-8-formatted text files. You can submit your documents in two formats. The format you use depends on the type of documents you want to analyze, as described below.

| Description | Format |
|---|---|
| Each file contains one input document. This is best for collections of large documents, such as newspaper articles or scientific papers. | One document per file |
| The input is one or more files. Each line in a file is considered a document. This is best for short documents, such as text messages or social media posts.<br><br>Each line must end with a line feed (LF, \n), a carriage return (CR, \r), or both (CRLF, \r\n). You can't use the UTF-8 line separator (u+2028) to end a line. | One document per line |

**One document per line**

With the `One document per line` method, all documents go in a single CSV-formatted text file. Each document is placed on a separate line and no header is used. The label is not included on each line (since you don't yet know the label for the document. Even though only the documents are present, one on each line, each entry must still conform to valid CSV format. Commas can be part of the document, for instance, but they must be escaped out. Additionally, each line of the file (end of the individual document) must end with \n.

The format of the input file can be seen as thus:

```
Text of document 1
Text of document 2
Text of document 3
Text of document 4
```

After preparing the documents file, you place that file in the S3 bucket that you're using for input data.

**One Document per File**

As with the previous method, the files used for this must be UTF-8 formatted text files. Each of thse is placed into the S3 bucket being used for input data.

The input data bucket contains the files used to run the classification job. Each file represents one document.

When you start a classification job, you will specify this S3 location for your input data. The URI must be in the same AWS Region as the API endpoint that you are calling. The URI can point to a single file (as when using the "one document per line" method, or it can be the prefix for a collection of data files.

For example, if you use the URI S3://bucketName/prefix, if the prefix is a single file, Amazon Comprehend uses that file as input. If more than one file begins with the prefix, Amazon Comprehend uses all of them as input.

You must grant Amazon Comprehend access to the Amazon S3 bucket that contains your document collection and output files. For more information, see Role-Based Permissions Required for Asynchronous Operations (p. 85).

## The Classification Job

Use the the section called “StartDocumentClassificationJob” (p. 174) operation to start classifying unlabeled documents. You provide the S3 bucket that contains the documents to be classified, the S3 bucket where the output should be placed, and classifier to use.

Custom classification is asynchronous. Once you have started the job, use the DescribeDocumentClassificationJob (p. 119) operation to monitor its progress. When the `Status` field in the response shows `COMPLETED`, you can access the output in the location that you specified.

The output is a single file named `output.tar.gz`. It is a compressed archive file that contain a text file with the output.

If you input was one document per line, the output file contains one line for each line in the input. Each line has the file name, the zero-based line number of the input line, the class label assigned to the document, and the confidence that Amazon Comprehend has that the file was correctly classified.

For example:

```
{"File": "file1.txt", "Line": "0", "Classes": [{"Name": "Cats", "Score": 0.8642}, {"Name":
 "Other", "Score": 0.0381}, {"Name": "Fluffy Animals", "Score": 0.0372}]}
{"File": "file1.txt", "Line": "1", "Classes": [{"Name": "Dogs", "Score": 0.5}, {"Name":
 "Dogs", "Score": 0.0381}, {"Name": "Dogs", "Score": 0.0372}]}
{"File": "file2.txt", "Line": "2", "Classes": [{"Name": "Cats", "Score": 0.1}, {"Name":
 "Cats", "Score": 0.0381}, {"Name": "Cats", "Score": 0.0372}]}
{"File": "file2.txt", "Line": "3", "Classes": [{"Name": "Fluffy Animals", "Score": 0.3141},
 {"Name": "Other", "Score": 0.0381}, {"Name": "Other", "Score": 0.0372}]}
```

If your input was one document per file, the output file contains one line for each document. Each line has the name of the file, the label of the class the file was assigned, and the confidence that Amazon Comprehend has that the file was correctly classified.

For example:

```
{"File": "file1.txt", "Classes": [{"Name": "Cats", "Score": 0.8642}, {"Name": "Other",
 "Score": 0.0381}, {"Name": "Fluffy Animals", "Score": 0.0372}]}
{"File": "file1.txt", "Classes": [{"Name": "Dogs", "Score": 0.5}, {"Name": "Dogs", "Score":
 0.0381}, {"Name": "Dogs", "Score": 0.0372}]}
{"File": "file2.txt", "Classes": [{"Name": "Cats", "Score": 0.1}, {"Name": "Cats", "Score":
 0.0381}, {"Name": "Cats", "Score": 0.0372}]}
{"File": "file2.txt", "Classes": [{"Name": "Fluffy Animals", "Score": 0.3141}, {"Name":
 "Other", "Score": 0.0381}, {"Name": "Other", "Score": 0.0372}]}
```

**Note**
For more information about the asynchronous analysis job format, see Asynchronous Batch Processing (p. 59)

# Custom Classifier Metrics

Amazon Comprehend provides you with metrics to help you estimate how well a custom classifier should work for your job. They are based on training the classifier model, and so while they accurately represent the performance of the model during training, they are only an approximation of the API performance during classification.

Metrics are included any time metadata from a trained custom classifier is returned.

**Note**
Please refer to Metrics: Precision, Recall, and FScore for an understanding of the underlying Precision, Recall, and F1 score metrics. These metrics are defined at a class level. We have used **macro** averaging for combining these metrics together to come up with the test set P,R,F1, as discussed below.

We support the following metrics:

- **Accuracy**
- **Macro Precision**
- **Macro Recall**
- **Macro F1 Score**

### Accuracy

Accuracy is reported at the corpus level. This indicates that it indicates the percentage of labels from the test data that are predicted exactly right by the model. This is also called *micro*-averaging.

For example

| Actual Label | Predicted Label | Right/Wrong |
|---|---|---|
| 1 | 1 | Right |
| 0 | 1 | Wrong |
| 2 | 3 | Wrong |
| 3 | 3 | Right |
| 2 | 2 | Right |
| 1 | 1 | Right |
| 3 | 3 | Right |

The accuracy consists of the number of "rights" / overall test samples = 5/7 = 0.714, or 71.4%

**Macro Precision, Macro Recall, Macro F1**

Macro Precision, Macro Recall, and Macro F1 are also defined at a corpus level. These calculate metrics for each label, and finds their unweighted mean. This does not take label imbalance into account.

- **Macro Precision**: Precision is defined as the number of documents correctly classified, over the total number of classifications for the class.. The macro average is macro-averaged per class precision.
- **Macro Recall**: Recall is defined as the number of classifications correctly identified over the same number plus the number of false negatives. Macro recall just averages the recall of all the classes together.
- **Macro F1**: F1 score is a harmonic mean of the Precision and Recall metrics. The Macro F1 score is the macro average of F1 scores of all the classes

This is demonstrated in the following example:

Given a test set with the following samples:

```
"label_1":400,"label_2":300, "label_3":30000,"label_4":20,"label_5":10,
```

If our classifier gets following :

```
Precision of "label_1": 0.75, label_2: 0.80, label_3: 0.90, label_4: 0.50, label_5: 0.40
Recall of "label_1": 0.70, label_2 : 0.70,label_3 :  0.98, label_4 : 0.80, label_5 : 0.10
F1 of "label_1": 0.724, label_2: 0.824, label_3: 0.94, label_4: 0.62, label_5: 0.16

Macro Precision = (0.75 + 0.80 + 0.90 + 0.50 + 0.40)/5 = 0.67
Macro Recall = (0.70 + 0.70 + 0.98 + 0.80 + 0.10)/5 = 0.656
Macro F1 = (0.724 + 0.824 + 0.94 + 0.62 + 0.16)/5 = 0.6536
```

Macro weighted values give equal weight to each of the classes, irrespective of their presence in the test data. A very uncommon class can have equal influence on final score as the most commonly occuring class, such as a class which is present in 75% of the examples.

One common usecase where macro averaging would give us more info over weighted metrics or the accuracy metric discussed above, is in case of extremely skewed datasets where the under-represented classes are very important. For example: in a cancer detection usecase where 95% of the samples would be negative, a very naive model which simply scores every sample as negative would score 95% on accuracy and 95% weighted precision, recall, and F1. But this wouldn't be very informative, where as a macro precision would compute this to be 50% because of its average of 1 for negative class and 0 for positive class. Note that the positive class is extremely important in this usecase.

## Improving Your Custom Classifier's Performance

The metrics provide an insight into how your custom classifier will perform during a classification job. If the metrics are low, it's very likely that the classification model might not be effective for your usecase. If this happens, you have several options to improve your classifier performance.

1. In your training data, provide more concrete data that can easily separate the categories. For example, provide documents that can best represent the label in terms of unique words/sentences.
2. Add more data for under-represented labels in your training data.
3. Try to reduce skew in the categories. If the largest label in your data is more than 10X the documents that are in the smallest label, try to increase the number of documents in the smallest and make sure to get the skew ratio down to at least 10:1 between highly represented and least represented classes. You can also try removing few documents from highly represented classes as well.

# Custom Entity Recognition

Custom entity recognition extends the capability of Amazon Comprehend by enabling you to identify new entity types not supported as one of the preset generic entity types. This means that in addition to identifying entity types such as LOCATION, DATE, PERSON, and so on, you can analyze documents and extract entities like product codes or business-specific entities that fit your particular needs.

Building an accurate in-house custom entity recognizer on your own can be a complex process, requiring preparation of large sets of manually annotated training documents and the selection of the right algorithms and parameters for model training. Amazon Comprehend enables you to sidestep some of these issues by providing automatic annotation and model development to create a custom entity recognition model.

Creating a custom entity recognition model is a more effective approach, compared to using string matching or regular expressions to identify entities. For example, to extract product codes, it would be difficult to enumerate all possible patterns to apply string matching. But a custom entity recognition model can learn the context where those product codes are most likely to appear and then make such inferences even though it has never previously seen the exact product codes. As well, typos in product codes and the addition of new product codes can still be expected to be caught by Amazon Comprehend's custom entity recognition model, but would be missed when using string matches against a static list.

To use Amazon Comprehend's custom entity recognition service, you need to provide a data set for model training purposes, with either a set of annotated documents, or a list of entities and their type label (such as `PRODUCT_CODES`) and a set of documents containing those entities. The service automatically tests for the best algorithm and parameters while training the model to use, looking for the most accurate combination of these components.

**Topics**



## Training Custom Entity Recognizers

Amazon Comprehend's custom entity recognition enables you to analyze your documents to find entities specific to your needs, rather than limiting you to the preset entity types already available. You can identity almost any kind of entity, simply by providing a sufficient number of details to train your model effectively.

Building a successful custom entity recognition model requires training your model. The training process usually requires extensive knowledge of machine learning (ML) and a complex process for model optimization. Amazon Comprehend automates this for you using a technique called *transfer learning* to build your model on a sophisticated general-purpose entities recognition model framework. With this in place, all you need to supply is the data. However, it's important that you supply it with high quality data as input. Without good data the model won't learn how to correctly identify entities.

You can choose one of two ways to provide data to Amazon Comprehend in order to train a custom entity recognition model:

- Annotations (p. 74)—This uses an annotation list that provides the location of your entities in a large number of documents so Amazon Comprehend can train on both the entity and its context.
- Entity Lists (p. 75)—This provides only a limited context, and uses only a list of the specific entities list so Amazon Comprehend can train to identify the custom entity.

**Annotations**

By submitting annotation along with your documents, you can increase the accuracy of the model. With Annotations, you're not simply providing the location of the entity you're looking for, but you're also providing more accurate context to the custom entity you're seeking.

For instance, if you're searching for the name John Johnson, with the entity type JUDGE, providing your annotation may help the model to learn that the person you want to find is a judge. If it is able to use the context, then Amazon Comprehend won't find people named John Johnson who are attorneys or witnesses. Without providing annotations, Amazon Comprehend will create its own version of an annotation, but won't be as effective at including only judges.

Providing your own annotation takes more work, but can be significantly more refined. Not using your own annotation is quicker and less expensive (in terms of work), but the results are rougher and less accurate.

**Entity Lists**

With custom entity recognition, Amazon Comprehend enables you to train your model without submitting annotation, by simply providing a list of the entities rather than an annotations file. This is a simpler and easier choice than the annotations option, but is probably going to result in a rougher, less specific result. This is because the annotations provide more context for Amazon Comprehend to use when training the model. Without that context, Amazon Comprehend will have a higher number of false positives when trying to identify the entities.

When you provide an Entity List, Amazon Comprehend uses an intelligent algorithm to detect occurrances of the entity in the documents to serve as the basis for training the custom entity recognizer model.

For instance, if you are searching for the name John Johnson, with the entity type JUDGE, using an entity list will enable you to identify instances of his name. However, without an annotations list to provide a clear context that will help Amazon Comprehend to determine if the instance is the correct John Johnson, it will identify instances where a person's name is John Johnson, but is not the correct individual.

**Annotations or Entity List?**

This comparison would make it seem as if an annotations list is always desirable, but this isn't the case. Only the name John Johnson might be significant to your search and whether it's the exact individual isn't relevant. Or the result is useful, but not worth the time and cost of producing an annotation list. Or the metrics when using the entity list are good enough to provide you with the results you need. There are many scenarios when it makes more business sense to avoid the higher expense and workload of creating the annotations necessary for the other option. In such instances, using an entity list instead can be the more effective choice.

We recommend using annotations in the following cases:

- When the meaning of the entities could be ambiguous and context-dependent. For example, the term *Amazon* could either refer to the river in Brazil, or the online retailer Amazon.com. When you build a custom entity recognizer to identify business entities such as *Amazon*, you should use annotations instead of an entity list because this method is better able to use context to find entities.
- When you are comfortable setting up a process to acquire annotations, which can require a significant amount of effort on your end.

We recommend using entity list in the following cases:

- When you already have a list of entities or when it is relatively easy to compose a comprehensive list of entities. If you use an entity list, the list should be complete or at least covers the majority of valid entities that may appear in the documents you provide for training. Do not provide a list with few

entities in it just to try out the service, because entities that appear in documents but not provided in the entity list will be treated as non-entities by Amazon Comprehend and may even hurt model training, especially if there are a lot of them.

- For first-time users, it is generally recommended to use an entity list because it involves less effort to prepare training data. It is important to note that the trained model might not be as accurate as if you used annotations.

**Data Options**

- Annotations (p. 74)
- Entity Lists (p. 75)

# Annotations

Annotations for custom entity recognition needs to be in a comma-separated value (CSV) file, with the following columns:

- **File**—The name of the containing the document. For example, if one of the document files is located at `s3://custom-ner-test-datasets-us-west-2/train-small-001/documents/file1`, the value in the `File` column should be `file1`. Please note that if the file name has an extension such as `.txt`, this must be included as part of the file name.
- **Line**—The line number containing the entity, starting with line 0.
- **Begin Offset**—The character offset in the input text (relative to the beginning of the line) that shows where the entity begins. The first character is at position 0.
- **End Offset**—The character offset in the input text that shows where the entity ends.
- **Type**—The customer-defined entity type. Entity types must an uppercase, underscore separated string such as JUDGE or FEDERAL_JUDGE. Only a single entity type can be trained per model.

For example:

The file `file2` contains two lines:

```
John Johnson is a judge on the Washington State Supreme Court.
Johnson has been a judge for 14 years.
```

The CSV file with the list of annotations would have the following lines:

```
File, Line, Begin Offset, End Offset, Type
file2, 0, 0, 11, JUDGE
file2, 1, 0, 6, JUDGE
```

> **Note**
> The annotations list must be a valid CSV file and also include the column names: `File, Line, Begin Offset, End Offset, Type`

A minimum of 1000 annotations are needed to train a model for custom entity recognition.

**Getting the Best Results**

There are a number of things to consider to get the best result when using annotations, inluding:

- Annotate your data with care. Imprecise annotations can lead to poor results.
- In general, more annotations will lead to better results.

- Input data should not contain duplicates. Presence of duplicate samples might result into test set contamination and therefore negatively affect training process, model metrics, and behavior.
- For best results, provide at least 50% coverage of document in corpus with annotations. Make sure that all documents in corpus have been annotated, and that the documents without annotations are due to lack of legitimate entities, not due to negligence. For example, if you have a document "Johnson has been a judge for 14 years", you should also provide an annotation for "Johnson". Failing to do so will confuse the model and might lead to not recognizing “Johnson” as JUDGE.
- Provide documents that resemble real usecases as closely as possible. Don't use toy data or synthesized data for production systems. The input data should be as diverse as possible to avoid overfitting and help underlying model better generalize on real examples.
- Try and give the same data distribution for training as you expect to be using when you're actually detecting your custom entitities (inference time). For example, at inference time, if you expect to be sending us documents that have no entities in them, this should also be part of your training document set.
- We recommend use open source web tools such as Brat Rapid Annotation Tool for data annotation, and then convert the annotations programmatically to fit the input format required by Amazon Comprehend. You can also search online for general guidelines on how to measure and improve the quality of human annotations.
- Amazon Mechanical Turk is a good resource for inexpensively annotating a large amount of data.

Additional suggestions can be found at Improving Custom Entity Recognizer Performance (p. 78)

## Entity Lists

An entity list for custom entity recognition needs a comma-separated value (CSV) file, with the following columns:

- **Text**— The text of an entry example exactly as seen in the accompanying document corpus.
- **Type**—The customer-defined entity type. Entity types must an uppercase, underscore separated string such as JUDGE or FEDERAL_JUDGE. Only a single entity type can be trained per model.

The file `file1` contains two lines:

```
John Johnson passed away shortly after birth.
Johnson is a judge on the Washington State Supreme Court.
```

The CSV file with the list of annotations would have the following lines:

```
Text, Type
John Johnson, JUDGE
Johnson, JUDGE
```

> **Note**
> The entity list must be a valid CSV file and also include the column names: `Text, Type`

**Getting the Best Results**

There are a number of things to consider to get the best result when using an entity list, inluding:

- The order of the entities in your list has no effects on model training.
- Use entity list items that cover 80%-100% of positive entity examples mentioned in the unannotated corpus of documents.
- Avoid entity examples that match non-entities in the document corpus by removing common words and phrases. Even a handful of incorrect matches can significantly affect the accuracy of your resulting

model. For example, a word like *the* in the entity list will result in a high number of matches which are unlikely to be the entities you are looking for and thus will significantly affect your accuracy.

- Input data should not contain duplicates. Presence of duplicate samples might result into test set contamination and therefore negatively affect training process, model metrics, and behavior.
- Provide documents that resemble real usecases as closely as possible. Don't use toy data or synthesized data for production systems. The input data should be as diverse as possible to avoid overfitting and help underlying model better generalize on real examples.
- The entity list is case sensitive, and regular expression are not currently supported. However, the trained model can often still recognize entities even if they do not match exactly to the casing provided in the entity list.
- If you have an entity that is a substring of another entity (eg “Johnson” and “John Johnson”), provide both in entity list.

Additional suggestions can be found at Improving Custom Entity Recognizer Performance (p. 78)

# Detecting Custom Entities

Once you have a trained model, use the StartEntitiesDetectionJob (p. 181) operation to detect custom entities in your documents.

Using this operation, you provide the same information as you would when detecting preset entities. However, in addition to the input and output locations (S3 buckets), you also provide the EntityRecognizerArn, which is the Amazon Resource Name (ARN) of the trained model. This ARN is supplied by the response to the CreateEntityRecognizer (p. 112) operation.

You can examine one document or many, but since each model is only trained on a single custom entity, you can only search for that one entity per StartEntitiesDetectionJob operation.

To detect custom entities in a document set, you use the following request syntax:

The examples are formatted for Unix, Linux, and macOS. For Windows, replace the backslash (\) Unix continuation character at the end of each line with a caret (^).

```
aws comprehend start-entities-detection-job \
     --entity-recognizer-arn "entity recognizer arn" \
     --job-name job name \
     --data-access-role-arn "data access role arn" \
     --language-code en \
     --input-data-config "S3Uri=s3://Bucket Name/Bucket Path" \
     --output-data-config "S3Uri=s3://Bucket Name/Bucket Path/" \
     --region region
```

Amazon Comprehend will respond with the `JobID` and `JobStatus` and will return the output from the job in the S3 bucket that you specified in your request. This output will be similar to the following:

```
{"File": "50_docs", "Line": 0, "Entities": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.9763959646224976, "Text": "John Johnson", "Type": "JUDGE"}"]}
{"File": "50_docs", "Line": 1, "Entities": [{"BeginOffset": 11, "EndOffset": 15, "Score":
 0.9615424871444702, "Text": "Thomas Kincaid", "Type": "JUDGE"}}]}
```

# Custom Entity Recognizer Metrics

Amazon Comprehend provides you with metrics to help you estimate how well an entity recognizer should work for your job. They are based on training the recognizer model, and so while they accurately

represent the performance of the model during training, they are only an approximation of the API performance during entity discovery.

Metrics are returned any time metadata from a trained entity recognizer is returned.

Three metrics are available:

- **Precision**
- **Recall**
- **F1 Score**

Currently, Amazon Comprehend does not support multiple custom entity types. These metrics are computed against the single custom entity type supported. As a result, we are not computing micro or macro precision/recall/f1 score.

**Precision**

This indicates how many times the model's entity identification is truly a good identification. It is a percentage of the total number of identifications.

Precision is defined as the number of entities correctly identified, over the total number of attempted identifications.

For example, say you are searching for city names in the following sentence:

```
John Smith lives in San Francisco, works in San Diego, and owns a house in Seattle.
```

The model returns "San Francisco" and "Smith", identified as cities from the sentence.

Precision is based on true positives (tp) and false positives (fp) and it is calculated as

precision = tp / (tp + fp).

In this case, the precision would be 50%, as one entity was correct out of the two identified by the model. One true positive and one false positive.

**Recall**

Recall is defined as the number of entities correctly identified (true positives) over the same number plus the number of false negative (fn) predictions. It is calculated as

recall = tp / (tp + fn)

In our example, searching for city names in the sentence:

```
John Smith lives in San Francisco, works in San Diego, and owns a house in Seattle.
```

The model returns "San Francisco" and "Smith", identified as cities from the sentence. This is one true positive predition ("San Francisco") and two false negatives ("San Diego", and "Seattle").

In this case, recall is the ratio of `1 / (1 + 2)`, or 33.33%, as one classification was a true positive, out of a possibility of three positives in the test set.

**F1 Score**

This is a combination of the Precision and Recall metrics, which measures the overall accuracy of the model for custom entity recognition. The F1 score is the harmonic mean of the Precision and Recall metrics:

F1 = 2 * Precision * Recall / (Precision + Recall)

> **Note**
> Intuitively, the harmonic mean penalizes the extremes more than the simple average or other means (example: `P` = 0, `R` = 1 could be achieved trivially by predicting all possible spans. Here, the simple average would be 0.5, but `F1` would penalize it as 0).

In the example above, `P` = 50% and `R` = 33.33%, therefore `F1` = 2 * 0.5 * 0.3333 / (0.5 + 0.3333).

The F1 Score is .3975, or 39.75%.

## Improving Custom Entity Recognizer Performance

These metrics provide an insight into how accurately the trained model will perform when you use it to identify entities. Here are a few options you can use to improve your metrics if they are lower than your expectations:

1. Depending on whether you use Annotations (p. 74) or Entity Lists (p. 75), make sure to follow the guidelines in the respective documentation to improve data quality. If you observe better metrics after improving your data and re-training the model, you can keep iterating and improving data quality to achieve better model performance.
2. If you are using an Entity List, consider using Annotations instead. The increased context of the annotations can often improve your metrics.
3. If you are sure there is not a data quality issue, and yet the metrics remain unreasonably low, please submit a support request.

# Authentication and Access Control for Amazon Comprehend

Access to Amazon Comprehend requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access Amazon Comprehend actions. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon Comprehend to help secure your resources by controlling who can access them.



## Authentication

You can access AWS as any of the following types of identities:

- **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 user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create in Amazon Comprehend). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon Comprehend supports *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*.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing user 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*.

- **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. 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 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*.

# Access Control

You must have valid credentials to authenticate your requests. The credentials must have permissions to call an Amazon Comprehend action.

The following sections describe how to manage permissions for Amazon Comprehend. We recommend that you read the overview first.



# Overview of Managing Access Permissions to Amazon Comprehend Resources

Permissions to access an action are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) to manage access to actions.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions and the actions they get permissions for.

**Topics**



## Managing Access to Actions

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

**Note**
This section discusses using IAM in the context of Amazon Comprehend. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM policies) and policies attached to a resource are referred to as *resource-based* policies. Amazon Comprehend supports only identity-based policies.

## Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user or a group of users permissions to call and Amazon Comprehend action, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – To grant cross-account permissions, you can attach an identity-based permissions policy to an IAM role. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
  2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. If you want to grant an AWS service permissions to assume the role, the principal in the trust policy can also be an AWS service principal.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

For more information about using identity-based policies with Amazon Comprehend, see Using Identity-Based Policies (IAM Policies) for Amazon Comprehend (p. 82). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

## Resource-Based Policies

Other services, such as Lambda, support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon Comprehend doesn't support resource-based policies.

# Specifying Policy Elements: Actions, Effects, and Principals

Amazon Comprehend defines a set of API operations (see Actions (p. 92)). To grant permissions for these API operations, Amazon Comprehend defines a set of actions that you can specify in a policy.

The following are the most basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies. For Amazon Comprehend, the resource is always `"*"`.

- **Action** – You use action keywords to identify operations that you want to allow or deny. For example, depending on the specified `Effect`, `comprehend:DetectEntities` either allows or denies the user permissions to perform the Amazon Comprehend `DetectEntities` operation.
- **Effect** – You specify the effect of the action that occurs when the user requests the specific action —this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource. You might do this to make sure that a user cannot access the resource, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table showing all of the Amazon Comprehend API actions, see Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference (p. 87).

## Specifying Conditions in a Policy

When you grant permissions, you use the IAM policy language to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

AWS provides a set of predefined condition keys for all AWS services that support IAM for access control. For example, you can use the `aws:userid` condition key to require a specific AWS ID when requesting an action. For more information and a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

> **Note**
> Condition keys are case sensitive.

Amazon Comprehend does not provide any additional condition keys.

# Using Identity-Based Policies (IAM Policies) for Amazon Comprehend

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform Amazon Comprehend actions.

> **Important**
> Before you proceed, we recommend that you review Overview of Managing Access Permissions to Amazon Comprehend Resources (p. 80).

The following is the permissions policy required to use the Amazon Comprehend document analysis actions:

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowDetectActions",
      "Effect": "Allow",
      "Action": [
                "comprehend:DetectEntities",
                "comprehend:DetectKeyPhrases",
```

```
                "comprehend:DetectDominantLanguage",
                "comprehend:DetectSentiment"
            ],
        "Resource": "*"
        }
    ]
}
```

The policy has one statement that grants permission to use the `DetectEntities`, `DetectKeyPhrases`, `DetectDominantLanguage` and `DetectSentiment` actions. A user with this policy would not be able to perform batch actions in your account.

The policy doesn't specify the `Principal` element because you don't specify the principal who gets the permission in an identity-based policy. When you attach a policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon Comprehend API actions and the resources that they apply to, see Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference (p. 87).

# Permissions Required to Use the Amazon Comprehend Console

The permissions reference table lists the Amazon Comprehend API operations and shows the required permissions for each operation. For more information, about Amazon Comprehend API permissions, see Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference (p. 87).

To use the Amazon Comprehend console, you need to grant permissions for the actions shown in the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "comprehend:*",
                "iam:ListRoles",
                "iam:GetRole",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

The Amazon Comprehend console needs these additional permissions for the following reasons:

- `iam` permissions to list the available IAM roles for your account.
- `s3` permissions to access the Amazon S3 buckets and objects that contain the data for topic modeling.

When you create an asynchronous batch job or a topic modeling job using the console, you have the option to have the console create an IAM role for your job. To create an IAM role, users must be granted the following additional permissions to create IAM roles and policies, and to attach policies to roles:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
      {
         "Action": [
            "iam:CreateRole",
            "iam:CreatePolicy",
            "iam:AttachRolePolicy",
            "iam:PassRole"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

The Amazon Comprehend console needs these additional permissions for the following reasons:

- `iam` permissions to create roles and policies and to attach roles and policies. The `iam:PassRole` action enables the console to pass the role to Amazon Comprehend.

# AWS Managed (Predefined) Policies for Amazon Comprehend

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Comprehend:

- **ComprehendFullAccess** – Grants full access to Amazon Comprehend resources including running topic modeling jobs. Includes permission to list and get IAM roles.
- **ComprehendReadOnly** – Grants permission to run all Amazon Comprehend actions except `StartDominantLanguageDetectionJob`, `StartEntitiesDetectionJob`, `StartKeyPhrasesDetectionJob`, `StartSentimentDetectionJob`, and `StartTopicsDetectionJob`.

You need to apply the following additional policy to any user that will use Amazon Comprehend:

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
         "Action": [
            "iam:PassRole"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

You can review the managed permissions policies by signing in to the IAM console and searching for specific policies there.

These policies work when you are using AWS SDKs or the AWS CLI.

You can also create your own custom IAM policies to allow permissions for Amazon Comprehend actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

# Role-Based Permissions Required for Asynchronous Operations

To use the Amazon Comprehend asynchronous operations, you must grant Amazon Comprehend access to the Amazon S3 bucket that contains your document collection. You do this by creating a data access role in your account to trust the Amazon Comprehend service principal. For more information about creating a role, see Creating a Role to Delegate Permissions to an AWS Service in the *AWS Identity and Access Management User Guide*.

The following is the role's trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "comprehend.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

After you have created the role, you must create an access policy for that role. The should grant the Amazon S3 `GetObject` and `ListBucket` permissions to the Amazon S3 bucket that contains your input data, and the Amazon S3 `PutObject` permission to your Amazon S3 output data bucket.

The following example access policy contains those permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::input bucket/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::input bucket"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::output bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

# Customer Managed Policy Examples

In this section, you can find example user policies that grant permissions for various Amazon Comprehend actions. These policies work when you are using AWS SDKs or the AWS CLI. When you are using the console, you need to grant permissions to all the Amazon Comprehend APIs. This is discussed in Permissions Required to Use the Amazon Comprehend Console (p. 83).

**Note**
All examples use the us-east-2 region and contain fictitious account IDs.

**Examples**

## Example 1: Allow All Amazon Comprehend Actions

After you sign up for AWS, you create an administrator user to manage your account, including creating users and managing their permissions.

You might choose to create a user who has permissions for all Amazon Comprehend actions (think of this user as a service-specific administrator) for working with Amazon Comprehend. You can attach the following permissions policy to this user.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowAllComprehendActions",
      "Effect": "Allow",
      "Action": [
         "comprehend:*"],
      "Resource": "*"
      }
   ]
}
```

## Example 2: Allow Topic Modeling Actions

The following permissions policy grants user permissions to perform the Amazon Comprehend topic modeling operations.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowTopicModelingActions",
      "Effect": "Allow",
      "Action": [
                "comprehend:DescribeTopicsDetectionJob",
                "comprehend:ListTopicsDetectionJobs",
                "comprehend:StartTopicsDetectionJob",
             ],
            "Resource": "*"
            ]
        }
    ]
```

```
}
```

# Amazon Comprehend API Permissions: Actions, Resources, and Conditions Reference

Use the following table as a reference when setting up Access Control (p. 80) and writing a permissions policy that you can attach to an IAM identity (an identity-based policy). The list includes each Amazon Comprehend API operation, the corresponding action for which you can grant permissions to perform the action, 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 Comprehend policies. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

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

# Guidelines and Limits

Keep in mind the following information when using Amazon Comprehend.

## Supported Regions

For a list of AWS Regions where Amazon Comprehend is availabe, see AWS Regions and Endpoints in the *Amazon Web Services General Reference*.

## Throttling

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

You may be able to avoid throttling by using the batch operations instead of the single transaction operations. For more information, see Multiple Document Operations (p. 88).

## Overall Limits

All operations except asynchronous operations and topic modeling operations have the following limits:

| Description | Limit |
|---|---|
| Character encoding | UTF-8 |
| Document size (UTF-8 characters) | 5,000 bytes |

Amazon Comprehend may store your content to continuously improve the quality of its analysis models. See the Amazon Comprehend FAQ to learn more. To request that we delete content that may have been stored by Amazon Comprehend, open a case with AWS Support.

## Multiple Document Operations

The BatchDetectDominantLanguage (p. 94), BatchDetectEntities (p. 97), BatchDetectKeyPhrases (p. 100), and BatchDetectSentiment (p. 103) operations have the following limits:

| Description | Limit |
|---|---|
| Documents per request | 25 |

If you plan to send more than 20 requests per second, you should consider using the batch operations. Batch operations enable you to send more documents in each request which may result in higher throughput. For example, when you use the `DetectDominantLanguage` operation, you can send up to 20 documents per second. However, if you use the `BatchRequestDominantLanguage` operation, you

can send up to 250 documents per second, but processing speed may be lower. For more information about throttling limits see Amazon Comprehend Limits in the *Amazon Web Services General Reference*. For more information about using the multiple document APIs, see Multiple Document Synchronous Processing (p. 63).

# Asynchronous Operations

Asynchronous batches started with the StartDominantLanguageDetectionJob (p. 178), StartEntitiesDetectionJob (p. 181), StartKeyPhrasesDetectionJob (p. 185), and StartSentimentDetectionJob (p. 188) have the following limits:

| Description | Limit |
|---|---|
| Maximum size (UTF-8 characters) for one document, entity and key phrase detection | 100 KB |
| Maximum size (UTF-8 characters) for one document, language detection | 1 MB |
| Maximum size (UTF-8 characters) for one document, sentiment detection | 5 KB |
| Total size of all files in batch | 5 Gb |
| Maximum number of files, one document per file | 1,000,000 |
| Maximum number of lines, one document per line | 1,000,000 |

You should use the asynchronous operations:

- To analyze more than 25 documents at a time
- To analyze documents larger than 5,000 bytes for keywords and entities

For more information, see Asynchronous Batch Processing (p. 59).

# Document Classification

Document classifier training jobs started with the CreateDocumentClassifier (p. 109) operation and document classification jobs started with the StartDocumentClassificationJob (p. 174) operation have the following limits:

| Description | Limit |
|---|---|
| Character encoding | UTF-8 |
| Maximum number of labels | 1,000 |
| Maximum length of labels | 5,000 characters |
| Total size of all files in request | 5 Gb |
| Maximum file size for one file, one document per file | 100 Mb |

| Description | Limit |
|---|---|
| Maximum number of files, one document per file | 1,000,000 |
| Maximum number of lines, one document per line | 1,000,000 |

# Language Detection

The BatchDetectDominantLanguage (p. 94), DetectDominantLanguage (p. 136) operations and asynchronous jobs started with the StartDominantLanguageDetectionJob (p. 178) operation have the following limitations:

- They don't support phonetic language detection. For example, they will not detect "arigato" as Japanese nor "nihao" as Chinese.
- They may have trouble distinguishing close language pairs, such as Indonesian and Malay; or Bosnian, Croatian, and Serbian.
- For best results the input text should be at least 20 characters long.

# Topic Modeling

Topic detection jobs created with the StartTopicsDetectionJob (p. 191) operation have the following limits:

| Description | Limit |
|---|---|
| Character encoding | UTF-8 |
| Maximum number of topics to return | 100 |
| Total size of all files in request | 5 Gb |
| Maximum file size for one file, one document per file | 100 Mb |
| Maximum number of files, one document per file | 1,000,000 |
| Maximum number of lines, one document per line | 1,000,000 |

For best results, you should include at least 1,000 input documents.

# Custom Entity Recognition

Custom entity recognition jobs started with the CreateEntityRecognizer (p. 112) operation have the following limits:

**General**

| Description | Minimum | Maximum |
|---|---|---|
| Number of entities per model/custom entity recognizer | -- | 1 |

| Description | Minimum | Maximum |
|---|---|---|
| Document size (UTF-8) | 1 byte | 5,000 |
| Number of documents | 2,000 | 120,000 |
| Document corpus size (all docs in plain text combined) | 5 KB | 100 MB |

### Annotations

| Description | Minimum | Maximum |
|---|---|---|
| Number of annotations | 1,000 | n/a |

### Entity Lists

| Description | Minimum | Maximum |
|---|---|---|
| Number of items in entity list | 1 | 1,000,000 |
| Length of individual entry (post-strip) in entry list | 1 | 5,000 |
| Entity list corpus size (all docs in plain text combined) | 5 KB | 100 MB |

# API Reference

This section provides documentation for the Amazon Comprehend API operations.

## Actions

The following actions are supported:

- StartTopicsDetectionJob (p. 191)
- StopDominantLanguageDetectionJob (p. 194)
- StopEntitiesDetectionJob (p. 196)
- StopKeyPhrasesDetectionJob (p. 198)
- StopSentimentDetectionJob (p. 200)

# BatchDetectDominantLanguage

Determines the dominant language of the input text for a batch of documents. For a list of languages that Amazon Comprehend can detect, see Amazon Comprehend Supported Languages.

## Request Syntax

```
{
   "TextList": [ "string" ]
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**TextList (p. 94)**

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document should contain at least 20 characters and must contain fewer than 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "Languages": [
            {
               "LanguageCode": "string",
               "Score": 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.

**ErrorList (p. 94)**

A list containing one BatchItemError (p. 208) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 208) objects

**ResultList (p. 94)**

A list of BatchDetectDominantLanguageItemResult (p. 203) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectDominantLanguageItemResult (p. 203) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**BatchSizeLimitExceededException**

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

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

# BatchDetectEntities

Inspects the text of a batch of documents for named entities and returns information about them. For more information about named entities, see Detect Entities (p. 50)

## Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 97)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**TextList (p. 97)**

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer than 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Entities": [
            {
               "BeginOffset": number,
               "EndOffset": number,
               "Score": number,
               "Text": "string",
```

```
                "Type": "string"
            }
        ],
        "Index": 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.

**ErrorList (p. 97)**

A list containing one BatchItemError (p. 208) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 208) objects

**ResultList (p. 97)**

A list of BatchDetectEntitiesItemResult (p. 204) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectEntitiesItemResult (p. 204) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**BatchSizeLimitExceededException**

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# BatchDetectKeyPhrases

Detects the key noun phrases found in a batch of documents.

## Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 100)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**TextList (p. 100)**

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "KeyPhrases": [
            {
               "BeginOffset": number,
               "EndOffset": number,
               "Score": number,
```

```
                "Text": "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.

**ErrorList (p. 100)**

A list containing one BatchItemError (p. 208) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 208) objects

**ResultList (p. 100)**

A list of BatchDetectKeyPhrasesItemResult (p. 205) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectKeyPhrasesItemResult (p. 205) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**BatchSizeLimitExceededException**

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# BatchDetectSentiment

Inspects a batch of documents and returns an inference of the prevailing sentiment, `POSITIVE`, `NEUTRAL`, `MIXED`, or `NEGATIVE`, in each one.

## Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 103)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**TextList (p. 103)**

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "Sentiment": "string",
         "SentimentScore": {
            "Mixed": number,
            "Negative": number,
```

```
            "Neutral": number,
            "Positive": 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.

**ErrorList (p. 103)**

A list containing one BatchItemError (p. 208) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 208) objects

**ResultList (p. 103)**

A list of BatchDetectSentimentItemResult (p. 206) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectSentimentItemResult (p. 206) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**BatchSizeLimitExceededException**

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# BatchDetectSyntax

Inspects the text of a batch of documents for the syntax and part of speech of the words in the document and returns information about them. For more information, see Analyze Syntax (p. 53).

## Request Syntax

```
{
   "LanguageCode": "string",
   "TextList": [ "string" ]
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 106)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es | fr | de | it | pt`

Required: Yes

**TextList (p. 106)**

A list containing the text of the input documents. The list can contain a maximum of 25 documents. Each document must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: Array of strings

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "ErrorList": [
      {
         "ErrorCode": "string",
         "ErrorMessage": "string",
         "Index": number
      }
   ],
   "ResultList": [
      {
         "Index": number,
         "SyntaxTokens": [
            {
               "BeginOffset": number,
               "EndOffset": number,
```

```
                "PartOfSpeech": {
                   "Score": number,
                   "Tag": "string"
                },
                "Text": "string",
                "TokenId": 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.

**ErrorList (p. 106)**

A list containing one BatchItemError (p. 208) object for each document that contained an error. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If there are no errors in the batch, the `ErrorList` is empty.

Type: Array of BatchItemError (p. 208) objects

**ResultList (p. 106)**

A list of BatchDetectSyntaxItemResult (p. 207) objects containing the results of the operation. The results are sorted in ascending order by the `Index` field and match the order of the documents in the input list. If all of the documents contain an error, the `ResultList` is empty.

Type: Array of BatchDetectSyntaxItemResult (p. 207) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**BatchSizeLimitExceededException**

The number of documents in the request exceeds the limit of 25. Try your request again with fewer documents.

HTTP Status Code: 400

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# CreateDocumentClassifier

Creates a new document classifier that you can use to categorize documents. To create a classifier you provide a set of training documents that labeled with the categories that you want to use. After the classifier is trained you can use it to categorize a set of labeled documents into the categories.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "DocumentClassifierName": "string",
   "InputDataConfig": {
      "S3Uri": "string"
   },
   "LanguageCode": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 109)**

A unique identifier for the request. If you don't set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 109)**

The Amazon Resource Name (ARN) of the AWS Identity and Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**DocumentClassifierName (p. 109)**

The name of the document classifier.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**InputDataConfig (p. 109)**

Specifies the format and location of the input data for the job.

Type: DocumentClassifierInputDataConfig (p. 215) object

Required: Yes

**LanguageCode (p. 109)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

## Response Syntax

```
{
   "DocumentClassifierArn": "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.

**DocumentClassifierArn (p. 110)**

The Amazon Resource Name (ARN) that identifies the document classifier.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:document-classifier/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceInUseException**

The specified name is already in use. Use a different name and try your request again.

HTTP Status Code: 400

**ResourceLimitExceededException**

The maximum number of recognizers per account has been exceeded. Review the recognizers, perform cleanup, and then try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# CreateEntityRecognizer

Creates an entity recognizer using submitted files. After your `CreateEntityRecognizer` request is submitted, you can check job status using the DescribeEntityRecognizer (p. 127) API.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "InputDataConfig": {
      "Annotations": {
         "S3Uri": "string"
      },
      "Documents": {
         "S3Uri": "string"
      },
      "EntityList": {
         "S3Uri": "string"
      },
      "EntityTypes": [
         {
            "Type": "string"
         }
      ]
   },
   "LanguageCode": "string",
   "RecognizerName": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 112)**

A unique identifier for the request. If you don't set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 112)**

The Amazon Resource Name (ARN) of the AWS Identity and Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**InputDataConfig (p. 112)**

Specifies the format and location of the input data. The S3 bucket containing the input data must be located in the same region as the entity recognizer being created.

Type: EntityRecognizerInputDataConfig (p. 232) object

Required: Yes

**LanguageCode (p. 112)**

The language of the input documents. All documents must be in the same language. Only English ("en") is currently supported.

Type: String

Valid Values: `en | es`

Required: Yes

**RecognizerName (p. 112)**

The name given to the newly created recognizer. Recognizer names can be a maximum of 256 characters. Alphanumeric characters, hyphens (-) and underscores (_) are allowed. The name must be unique in the account/region.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

## Response Syntax

```
{
   "EntityRecognizerArn": "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.

**EntityRecognizerArn (p. 113)**

The Amazon Resource Name (ARN) that identifies the entity recognizer.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:entity-recognizer/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceInUseException**

The specified name is already in use. Use a different name and try your request again.

HTTP Status Code: 400

**ResourceLimitExceededException**

The maximum number of recognizers per account has been exceeded. Review the recognizers, perform cleanup, and then try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# DeleteDocumentClassifier

Deletes a previously created document classifier

Only those classifiers that are in terminated states (IN_ERROR, TRAINED) will be deleted. If an active inference job is using the model, a `ResourceInUseException` will be returned.

This is an asynchronous action that puts the classifier into a DELETING state, and it is then removed by a background job. Once removed, the classifier disappears from your account and is no longer available for use.

## Request Syntax

```
{
   "DocumentClassifierArn": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**DocumentClassifierArn (p. 115)**

The Amazon Resource Name (ARN) that identifies the document classifier.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:document-classifier/[a-zA-Z0-9](-*[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. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceInUseException**

The specified name is already in use. Use a different name and try your request again.

HTTP Status Code: 400

**ResourceNotFoundException**

The specified resource ARN was not found. Check the ARN and try your request again.

HTTP Status Code: 400

**ResourceUnavailableException**

The specified resource is not available. Check to see if the resource is in the `TRAINED` state and try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DeleteEntityRecognizer

Deletes an entity recognizer.

Only those recognizers that are in terminated states (IN_ERROR, TRAINED) will be deleted. If an active inference job is using the model, a `ResourceInUseException` will be returned.

This is an asynchronous action that puts the recognizer into a DELETING state, and it is then removed by a background job. Once removed, the recognizer disappears from your account and is no longer available for use.

## Request Syntax

```
{
   "EntityRecognizerArn": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**EntityRecognizerArn (p. 117)**

The Amazon Resource Name (ARN) that identifies the entity recognizer.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:entity-recognizer/[a-zA-Z0-9](-*[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. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceInUseException**

The specified name is already in use. Use a different name and try your request again.

HTTP Status Code: 400

**ResourceNotFoundException**

The specified resource ARN was not found. Check the ARN and try your request again.

HTTP Status Code: 400

**ResourceUnavailableException**

The specified resource is not available. Check to see if the resource is in the `TRAINED` state and try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeDocumentClassificationJob

Gets the properties associated with a document classification job. Use this operation to get the status of a classification job.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 119)**

The identifier that Amazon Comprehend generated for the job. The StartDocumentClassificationJob (p. 174) operation returns this identifier in its response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "DocumentClassificationJobProperties": {
      "DataAccessRoleArn": "string",
      "DocumentClassifierArn": "string",
      "EndTime": number,
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "Message": "string",
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": 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.

**DocumentClassificationJobProperties (p. 119)**

An object that describes the properties associated with the document classification job.

Type: DocumentClassificationJobProperties (p. 212) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeDocumentClassifier

Gets the properties associated with a document classifier.

## Request Syntax

```
{
   "DocumentClassifierArn": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**DocumentClassifierArn (p. 121)**

The Amazon Resource Name (ARN) that identifies the document classifier. The CreateDocumentClassifier (p. 109) operation returns this identifier in its response.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:document-classifier/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

## Response Syntax

```
{
   "DocumentClassifierProperties": {
      "ClassifierMetadata": {
         "EvaluationMetrics": {
            "Accuracy": number,
            "F1Score": number,
            "Precision": number,
            "Recall": number
         },
         "NumberOfLabels": number,
         "NumberOfTestDocuments": number,
         "NumberOfTrainedDocuments": number
      },
      "DataAccessRoleArn": "string",
      "DocumentClassifierArn": "string",
      "EndTime": number,
      "InputDataConfig": {
         "S3Uri": "string"
      },
      "LanguageCode": "string",
      "Message": "string",
      "Status": "string",
      "SubmitTime": number,
      "TrainingEndTime": number,
      "TrainingStartTime": 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.

**DocumentClassifierProperties (p. 121)**

An object that contains the properties associated with a document classifier.

Type: DocumentClassifierProperties (p. 216) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceNotFoundException**

The specified resource ARN was not found. Check the ARN and try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeDominantLanguageDetectionJob

Gets the properties associated with a dominant language detection job. Use this operation to get the status of a detection job.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 123)**

The identifier that Amazon Comprehend generated for the job. The StartDominantLanguageDetectionJob (p. 178) operation returns this identifier in its response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "DominantLanguageDetectionJobProperties": {
      "DataAccessRoleArn": "string",
      "EndTime": number,
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "Message": "string",
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": 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.

**DominantLanguageDetectionJobProperties (p. 123)**

An object that contains the properties associated with a dominant language detection job.

Type: DominantLanguageDetectionJobProperties (p. 220) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeEntitiesDetectionJob

Gets the properties associated with an entities detection job. Use this operation to get the status of a detection job.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 125)**

The identifier that Amazon Comprehend generated for the job. The StartEntitiesDetectionJob (p. 181) operation returns this identifier in its response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "EntitiesDetectionJobProperties": {
      "DataAccessRoleArn": "string",
      "EndTime": number,
      "EntityRecognizerArn": "string",
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "LanguageCode": "string",
      "Message": "string",
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": 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.

**EntitiesDetectionJobProperties (p. 125)**

An object that contains the properties associated with an entities detection job.

Type: EntitiesDetectionJobProperties (p. 223) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeEntityRecognizer

Provides details about an entity recognizer including status, S3 buckets containing training data, recognizer metadata, metrics, and so on.

## Request Syntax

```
{
   "EntityRecognizerArn": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**EntityRecognizerArn (p. 127)**

The Amazon Resource Name (ARN) that identifies the entity recognizer.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:entity-recognizer/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

## Response Syntax

```
{
   "EntityRecognizerProperties": {
      "DataAccessRoleArn": "string",
      "EndTime": number,
      "EntityRecognizerArn": "string",
      "InputDataConfig": {
         "Annotations": {
            "S3Uri": "string"
         },
         "Documents": {
            "S3Uri": "string"
         },
         "EntityList": {
            "S3Uri": "string"
         },
         "EntityTypes": [
            {
               "Type": "string"
            }
         ]
      },
      "LanguageCode": "string",
      "Message": "string",
      "RecognizerMetadata": {
         "EntityTypes": [
            {
```

```
                "Type": "string"
            }
         ],
         "EvaluationMetrics": {
            "F1Score": number,
            "Precision": number,
            "Recall": number
         },
         "NumberOfTestDocuments": number,
         "NumberOfTrainedDocuments": number
      },
      "Status": "string",
      "SubmitTime": number,
      "TrainingEndTime": number,
      "TrainingStartTime": 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.

**EntityRecognizerProperties (p. 127)**

Describes information associated with an entity recognizer.

Type: EntityRecognizerProperties (p. 235) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceNotFoundException**

The specified resource ARN was not found. Check the ARN and try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeKeyPhrasesDetectionJob

Gets the properties associated with a key phrases detection job. Use this operation to get the status of a detection job.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 130)**

The identifier that Amazon Comprehend generated for the job. The StartKeyPhrasesDetectionJob (p. 185) operation returns this identifier in its response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "KeyPhrasesDetectionJobProperties": {
      "DataAccessRoleArn": "string",
      "EndTime": number,
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "LanguageCode": "string",
      "Message": "string",
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": 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.

**KeyPhrasesDetectionJobProperties (p. 130)**

An object that contains the properties associated with a key phrases detection job.

Type: KeyPhrasesDetectionJobProperties (p. 241) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeSentimentDetectionJob

Gets the properties associated with a sentiment detection job. Use this operation to get the status of a detection job.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 132)**

The identifier that Amazon Comprehend generated for the job. The StartSentimentDetectionJob (p. 188) operation returns this identifier in its response.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "SentimentDetectionJobProperties": {
      "DataAccessRoleArn": "string",
      "EndTime": number,
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "LanguageCode": "string",
      "Message": "string",
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": 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.

**SentimentDetectionJobProperties (p. 132)**

An object that contains the properties associated with a sentiment detection job.

Type: SentimentDetectionJobProperties (p. 246) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DescribeTopicsDetectionJob

Gets the properties associated with a topic detection job. Use this operation to get the status of a detection job.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 134)**

The identifier assigned by the user to the detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "TopicsDetectionJobProperties": {
      "EndTime": number,
      "InputDataConfig": {
         "InputFormat": "string",
         "S3Uri": "string"
      },
      "JobId": "string",
      "JobName": "string",
      "JobStatus": "string",
      "Message": "string",
      "NumberOfTopics": number,
      "OutputDataConfig": {
         "S3Uri": "string"
      },
      "SubmitTime": 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.

**TopicsDetectionJobProperties (p. 134)**

The list of properties for the requested job.

Type: TopicsDetectionJobProperties (p. 251) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# DetectDominantLanguage

Determines the dominant language of the input text. For a list of languages that Amazon Comprehend can detect, see Amazon Comprehend Supported Languages.

## Request Syntax

```
{
   "Text": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Text (p. 136)**

A UTF-8 text string. Each string should contain at least 20 characters and must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "Languages": [
      {
         "LanguageCode": "string",
         "Score": 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.

**Languages (p. 136)**

The languages that Amazon Comprehend detected in the input text. For each language, the response returns the RFC 5646 language code and the level of confidence that Amazon Comprehend has in the accuracy of its inference. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

Type: Array of DominantLanguage (p. 218) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

## Example

### Detect dominant language

If the input text is "Bob lives in Seattle. He is a software engineer at Amazon.", the operation returns the following:

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9774383902549744
        },
        {
            "LanguageCode": "de",
            "Score": 0.010717987082898617
        }
    ]
}
```

## 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

# DetectEntities

Inspects text for named entities, and returns information about them. For more information, about named entities, see Detect Entities (p. 50).

## Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 139)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**Text (p. 139)**

A UTF-8 text string. Each string must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "Entities": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "Score": number,
         "Text": "string",
         "Type": "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.

**Entities (p. 139)**

A collection of entities identified in the input text. For each entity, the response provides the entity text, entity type, where the entity text begins and ends, and the level of confidence that Amazon Comprehend has in the detection. For a list of entity types, see Detect Entities (p. 50).

Type: Array of Entity (p. 225) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

HTTP Status Code: 400

## Example

### Detect entities

If the input text is "Bob ordered two sandwiches and three ice cream cones today from a store in Seattle.", the operation returns the following:

```
{
"Entities": [
    {
        "Text": "Bob",
        "Score": 1.0,
        "Type": "PERSON",
        "BeginOffset": 0,
        "EndOffset": 3
    },
    {
        "Text": "two",
        "Score": 1.0,
        "Type": "QUANTITY",
```

```
            "BeginOffset": 12,
            "EndOffset": 15
        },
        {
            "Text": "three",
            "Score": 1.0,
            "Type": "QUANTITY",
            "BeginOffset": 32,
            "EndOffset": 37
        },
        {
            "Text": "Today",
            "Score": 1.0,
            "Type": "DATE",
            "BeginOffset": 54,
            "EndOffset": 59
        },
        {
            "Text": "Seattle",
            "Score": 1.0,
            "Type": "LOCATION",
            "BeginOffset": 76,
            "EndOffset": 83
        }
    ],
}
```


## 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

# DetectKeyPhrases

Detects the key noun phrases found in the text.

## Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 142)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**Text (p. 142)**

A UTF-8 text string. Each string must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "KeyPhrases": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "Score": number,
         "Text": "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.

**KeyPhrases (p. 142)**

A collection of key phrases that Amazon Comprehend identified in the input text. For each key phrase, the response provides the text of the key phrase, where the key phrase begins and ends, and the level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Array of KeyPhrase (p. 239) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

HTTP Status Code: 400

## Example

### Detect phrases

If the input text is "Bob lives in Seattle. He is a software engineer at Amazon.", the API returns the following:

```
      {
"KeyPhrases": [
    {
        "Text": "Bob",
        "Score": 1.0,
        "BeginOffset": 0,
        "EndOffset": 3
    },
    {
        "Text": "Seattle",
        "Score": 1.0,
        "BeginOffset": 13,
        "EndOffset": 20
    },
    {
```

```
            "Text": "an engineer",
            "Score": 1.0,
            "BeginOffset": 28,
            "EndOffset": 39
        },
        {
            "Text": "Amazon",
            "Score": 1.0,
            "BeginOffset": 43,
            "EndOffset": 49
        }
    ]
}}
```

## 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

# DetectSentiment

Inspects text and returns an inference of the prevailing sentiment (`POSITIVE`, `NEUTRAL`, `MIXED`, or `NEGATIVE`).

## Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 145)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**Text (p. 145)**

A UTF-8 text string. Each string must contain fewer that 5,000 bytes of UTF-8 encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "Sentiment": "string",
   "SentimentScore": {
      "Mixed": number,
      "Negative": number,
      "Neutral": number,
      "Positive": 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.

**Sentiment (p. 145)**

The inferred sentiment that Amazon Comprehend has the highest level of confidence in.

Type: String

Valid Values: `POSITIVE | NEGATIVE | NEUTRAL | MIXED`

**SentimentScore (p. 145)**

An object that lists the sentiments, and their corresponding confidence levels.

Type: SentimentScore (p. 248) object

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

HTTP Status Code: 400

## Example

### Detect sentiment

If the input text is "Today is my birthday, I am so happy.", the operation returns the following response:

```
{
    "SentimentScore": {
        "Mixed": 0.0033542951568961143,
        "Positive": 0.9869875907897949,
        "Neutral": 0.008563132025301456,
        "Negative": 0.0010949420975521207
    },
    "Sentiment": "POSITIVE",
 }
}
```

## 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

# DetectSyntax

Inspects text for syntax and the part of speech of words in the document. For more information, Analyze Syntax (p. 53).

## Request Syntax

```
{
   "LanguageCode": "string",
   "Text": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**LanguageCode (p. 148)**

The language code of the input documents. You can specify English ("en") or Spanish ("es").

Type: String

Valid Values: `en | es | fr | de | it | pt`

Required: Yes

**Text (p. 148)**

A UTF-8 string. Each string must contain fewer that 5,000 bytes of UTF encoded characters.

Type: String

Length Constraints: Minimum length of 1.

Required: Yes

## Response Syntax

```
{
   "SyntaxTokens": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "PartOfSpeech": {
            "Score": number,
            "Tag": "string"
         },
         "Text": "string",
         "TokenId": 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.

**SyntaxTokens (p. 148)**

A collection of syntax tokens describing the text. For each token, the response provides the text, the token type, where the text begins and ends, and the level of confidence that Amazon Comprehend has that the token is correct. For a list of token types, see Analyze Syntax (p. 53).

Type: Array of SyntaxToken (p. 249) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TextSizeLimitExceededException**

The size of the input text exceeds the limit. Use a smaller document.

HTTP Status Code: 400

**UnsupportedLanguageException**

Amazon Comprehend can't process the language of the input text. For all custom entity recognition APIs (such as `CreateEntityRecognizer`), only English is accepted. For most other APIs, Amazon Comprehend accepts only English or Spanish text.

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

# ListDocumentClassificationJobs

Gets a list of the documentation classification jobs that you have submitted.

## Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 150)**

Filters the jobs that are returned. You can filter jobs on their names, status, or the date and time that they were submitted. You can only set one filter at a time.

Type: DocumentClassificationJobFilter (p. 211) object

Required: No

**MaxResults (p. 150)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 150)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "DocumentClassificationJobPropertiesList": [
      {
```

```
         "DataAccessRoleArn": "string",
         "DocumentClassifierArn": "string",
         "EndTime": number,
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "Message": "string",
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": number
      }
   ],
   "NextToken": "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.

**DocumentClassificationJobPropertiesList (p. 150)**

A list containing the properties of each job returned.

Type: Array of DocumentClassificationJobProperties (p. 212) objects

**NextToken (p. 150)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListDocumentClassifiers

Gets a list of the document classifiers that you have created.

## Request Syntax

```
{
   "Filter": {
      "Status": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 153)**

Filters the jobs that are returned. You can filter jobs on their name, status, or the date and time that they were submitted. You can only set one filter at a time.

Type: DocumentClassifierFilter (p. 214) object

Required: No

**MaxResults (p. 153)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 153)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "DocumentClassifierPropertiesList": [
      {
         "ClassifierMetadata": {
            "EvaluationMetrics": {
```

```
                "Accuracy": number,
                "F1Score": number,
                "Precision": number,
                "Recall": number
             },
             "NumberOfLabels": number,
             "NumberOfTestDocuments": number,
             "NumberOfTrainedDocuments": number
          },
          "DataAccessRoleArn": "string",
          "DocumentClassifierArn": "string",
          "EndTime": number,
          "InputDataConfig": {
             "S3Uri": "string"
          },
          "LanguageCode": "string",
          "Message": "string",
          "Status": "string",
          "SubmitTime": number,
          "TrainingEndTime": number,
          "TrainingStartTime": number
       }
   ],
   "NextToken": "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.

**DocumentClassifierPropertiesList (p. 153)**

A list containing the properties of each job returned.

Type: Array of DocumentClassifierProperties (p. 216) objects

**NextToken (p. 153)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListDominantLanguageDetectionJobs

Gets a list of the dominant language detection jobs that you have submitted.

## Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 156)**

Filters that jobs that are returned. You can filter jobs on their name, status, or the date and time that they were submitted. You can only set one filter at a time.

Type: DominantLanguageDetectionJobFilter (p. 219) object

Required: No

**MaxResults (p. 156)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 156)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "DominantLanguageDetectionJobPropertiesList": [
      {
         "DataAccessRoleArn": "string",
```

```
         "EndTime": number,
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "Message": "string",
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": number
      }
   ],
   "NextToken": "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.

**DominantLanguageDetectionJobPropertiesList (p. 156)**

A list containing the properties of each job that is returned.

Type: Array of DominantLanguageDetectionJobProperties (p. 220) objects

**NextToken (p. 156)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListEntitiesDetectionJobs

Gets a list of the entity detection jobs that you have submitted.

## Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 159)**

Filters the jobs that are returned. You can filter jobs on their name, status, or the date and time that they were submitted. You can only set one filter at a time.

Type: EntitiesDetectionJobFilter (p. 222) object

Required: No

**MaxResults (p. 159)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 159)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "EntitiesDetectionJobPropertiesList": [
      {
         "DataAccessRoleArn": "string",
```

```
         "EndTime": number,
         "EntityRecognizerArn": "string",
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "LanguageCode": "string",
         "Message": "string",
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": number
      }
   ],
   "NextToken": "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.

**EntitiesDetectionJobPropertiesList (p. 159)**

A list containing the properties of each job that is returned.

Type: Array of EntitiesDetectionJobProperties (p. 223) objects

**NextToken (p. 159)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListEntityRecognizers

Gets a list of the properties of all entity recognizers that you created, including recognizers currently in training. Allows you to filter the list of recognizers based on criteria such as status and submission time. This call returns up to 500 entity recognizers in the list, with a default number of 100 recognizers in the list.

The results of this list are not in any particular order. Please get the list and sort locally if needed.

## Request Syntax

```
{
   "Filter": {
      "Status": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 162)**

Filters the list of entities returned. You can filter on `Status`, `SubmitTimeBefore`, or `SubmitTimeAfter`. You can only set one filter at a time.

Type: EntityRecognizerFilter (p. 231) object

Required: No

**MaxResults (p. 162)**

The maximum number of results to return on each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 162)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
```

```
   "EntityRecognizerPropertiesList": [
      {
         "DataAccessRoleArn": "string",
         "EndTime": number,
         "EntityRecognizerArn": "string",
         "InputDataConfig": {
            "Annotations": {
               "S3Uri": "string"
            },
            "Documents": {
               "S3Uri": "string"
            },
            "EntityList": {
               "S3Uri": "string"
            },
            "EntityTypes": [
               {
                  "Type": "string"
               }
            ]
         },
         "LanguageCode": "string",
         "Message": "string",
         "RecognizerMetadata": {
            "EntityTypes": [
               {
                  "Type": "string"
               }
            ],
            "EvaluationMetrics": {
               "F1Score": number,
               "Precision": number,
               "Recall": number
            },
            "NumberOfTestDocuments": number,
            "NumberOfTrainedDocuments": number
         },
         "Status": "string",
         "SubmitTime": number,
         "TrainingEndTime": number,
         "TrainingStartTime": number
      }
   ],
   "NextToken": "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.

**EntityRecognizerPropertiesList (p. 162)**

The list of properties of an entity recognizer.

Type: Array of EntityRecognizerProperties (p. 235) objects

**NextToken (p. 162)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListKeyPhrasesDetectionJobs

Get a list of key phrase detection jobs that you have submitted.

## Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 165)**

Filters the jobs that are returned. You can filter jobs on their name, status, or the date and time that they were submitted. You can only set one filter at a time.

Type: KeyPhrasesDetectionJobFilter (p. 240) object

Required: No

**MaxResults (p. 165)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 165)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "KeyPhrasesDetectionJobPropertiesList": [
      {
```

```
         "DataAccessRoleArn": "string",
         "EndTime": number,
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "LanguageCode": "string",
         "Message": "string",
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": number
      }
   ],
   "NextToken": "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.

**KeyPhrasesDetectionJobPropertiesList (p. 165)**

A list containing the properties of each job that is returned.

Type: Array of KeyPhrasesDetectionJobProperties (p. 241) objects

**NextToken (p. 165)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListSentimentDetectionJobs

Gets a list of sentiment detection jobs that you have submitted.

## Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 168)**

Filters the jobs that are returned. You can filter jobs on their name, status, or the date and time that they were submitted. You can only set one filter at a time.

Type: SentimentDetectionJobFilter (p. 245) object

Required: No

**MaxResults (p. 168)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 168)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "NextToken": "string",
   "SentimentDetectionJobPropertiesList": [
```

```
      {
         "DataAccessRoleArn": "string",
         "EndTime": number,
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "LanguageCode": "string",
         "Message": "string",
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": 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. 168)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

**SentimentDetectionJobPropertiesList (p. 168)**

A list containing the properties of each job that is returned.

Type: Array of SentimentDetectionJobProperties (p. 246) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# ListTopicsDetectionJobs

Gets a list of the topic detection jobs that you have submitted.

## Request Syntax

```
{
   "Filter": {
      "JobName": "string",
      "JobStatus": "string",
      "SubmitTimeAfter": number,
      "SubmitTimeBefore": number
   },
   "MaxResults": number,
   "NextToken": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**Filter (p. 171)**

Filters the jobs that are returned. Jobs can be filtered on their name, status, or the date and time that they were submitted. You can set only one filter at a time.

Type: TopicsDetectionJobFilter (p. 250) object

Required: No

**MaxResults (p. 171)**

The maximum number of results to return in each page. The default is 100.

Type: Integer

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

Required: No

**NextToken (p. 171)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

Required: No

## Response Syntax

```
{
   "NextToken": "string",
   "TopicsDetectionJobPropertiesList": [
      {
```

```
         "EndTime": number,
         "InputDataConfig": {
            "InputFormat": "string",
            "S3Uri": "string"
         },
         "JobId": "string",
         "JobName": "string",
         "JobStatus": "string",
         "Message": "string",
         "NumberOfTopics": number,
         "OutputDataConfig": {
            "S3Uri": "string"
         },
         "SubmitTime": 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. 171)**

Identifies the next page of results to return.

Type: String

Length Constraints: Minimum length of 1.

**TopicsDetectionJobPropertiesList (p. 171)**

A list containing the properties of each job that is returned.

Type: Array of TopicsDetectionJobProperties (p. 251) objects

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidFilterException**

The filter specified for the `ListDocumentClassificationJobs` operation is invalid. Specify a different filter.

HTTP Status Code: 400

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StartDocumentClassificationJob

Starts an asynchronous document classification job. Use the DescribeDocumentClassificationJob (p. 119) operation to track the progress of the job.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "DocumentClassifierArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 174)**

A unique identifier for the request. If you do not set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 174)**

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**DocumentClassifierArn (p. 174)**

The Amazon Resource Name (ARN) of the document classifier to use to process the job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:document-classifier/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**InputDataConfig (p. 174)**

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 238) object

Required: Yes

**JobName (p. 174)**

The identifier of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**OutputDataConfig (p. 174)**

Specifies where to send the output files.

Type: OutputDataConfig (p. 243) object

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 175)**

The identifier generated for the job. To get the status of the job, use this identifier with the DescribeDocumentClassificationJob (p. 119) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 175)**

The status of the job:

- SUBMITTED - The job has been received and queued for processing.

- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. For details, use the DescribeDocumentClassificationJob (p. 119) operation.
- STOP_REQUESTED - Amazon Comprehend has received a stop request for the job and is processing the request.
- STOPPED - The job was successfully stopped without completing.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceNotFoundException**

The specified resource ARN was not found. Check the ARN and try your request again.

HTTP Status Code: 400

**ResourceUnavailableException**

The specified resource is not available. Check to see if the resource is in the `TRAINED` state and try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StartDominantLanguageDetectionJob

Starts an asynchronous dominant language detection job for a collection of documents. Use the DescribeDominantLanguageDetectionJob (p. 123) operation to track the status of a job.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 178)**

A unique identifier for the request. If you do not set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 178)**

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data. For more information, see https://docs.aws.amazon.com/comprehend/latest/dg/access-control-managing-permissions.html#auth-role-permissions.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**InputDataConfig (p. 178)**

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 238) object

Required: Yes

**JobName (p. 178)**

An identifier for the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**OutputDataConfig (p. 178)**

Specifies where to send the output files.

Type: OutputDataConfig (p. 243) object

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 179)**

The identifier generated for the job. To get the status of a job, use this identifier with the DescribeDominantLanguageDetectionJob (p. 123) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 179)**

The status of the job.

- SUBMITTED - The job has been received and is queued for processing.
- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. To get details, use the DescribeDominantLanguageDetectionJob (p. 123) operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StartEntitiesDetectionJob

Starts an asynchronous entity detection job for a collection of documents. Use the DescribeEntitiesDetectionJob (p. 125) operation to track the status of a job.

This API can be used for either standard entity detection or custom entity recognition. In order to be used for custom entity recognition, the optional `EntityRecognizerArn` must be used in order to provide access to the recognizer being used to detect the custom entity.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "EntityRecognizerArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "LanguageCode": "string",
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 181)**

A unique identifier for the request. If you don't set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 181)**

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data. For more information, see https://docs.aws.amazon.com/comprehend/latest/dg/access-control-managing-permissions.html#auth-role-permissions.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**EntityRecognizerArn (p. 181)**

The Amazon Resource Name (ARN) that identifies the specific entity recognizer to be used by the `StartEntitiesDetectionJob`. This ARN is optional and is only used for a custom entity recognition job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:entity-recognizer/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**InputDataConfig (p. 181)**

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 238) object

Required: Yes

**JobName (p. 181)**

The identifier of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**LanguageCode (p. 181)**

The language of the input documents. All documents must be in the same language. You can specify any of the languages supported by Amazon Comprehend: English ("en"), Spanish ("es"), French ("fr"), German ("de"), Italian ("it"), or Portuguese ("pt"). If custom entities recognition is used, this parameter is ignored and the language used for training the model is used instead.

Type: String

Valid Values: `en | es`

Required: Yes

**OutputDataConfig (p. 181)**

Specifies where to send the output files.

Type: OutputDataConfig (p. 243) object

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 182)**

The identifier generated for the job. To get the status of job, use this identifier with the DescribeEntitiesDetectionJob (p. 125) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 182)**

The status of the job.

- SUBMITTED - The job has been received and is queued for processing.
- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. To get details, use the DescribeEntitiesDetectionJob (p. 125) operation.
- STOP_REQUESTED - Amazon Comprehend has received a stop request for the job and is processing the request.
- STOPPED - The job was successfully stopped without completing.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**ResourceNotFoundException**

The specified resource ARN was not found. Check the ARN and try your request again.

HTTP Status Code: 400

**ResourceUnavailableException**

The specified resource is not available. Check to see if the resource is in the `TRAINED` state and try your request again.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StartKeyPhrasesDetectionJob

Starts an asynchronous key phrase detection job for a collection of documents. Use the DescribeKeyPhrasesDetectionJob (p. 130) operation to track the status of a job.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "LanguageCode": "string",
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 185)**

A unique identifier for the request. If you don't set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 185)**

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data. For more information, see https://docs.aws.amazon.com/comprehend/latest/dg/access-control-managing-permissions.html#auth-role-permissions.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**InputDataConfig (p. 185)**

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 238) object

Required: Yes

**JobName (p. 185)**

The identifier of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**LanguageCode (p. 185)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**OutputDataConfig (p. 185)**

Specifies where to send the output files.

Type: OutputDataConfig (p. 243) object

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 186)**

The identifier generated for the job. To get the status of a job, use this identifier with the DescribeKeyPhrasesDetectionJob (p. 130) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 186)**

The status of the job.

- SUBMITTED - The job has been received and is queued for processing.

- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. To get details, use the DescribeKeyPhrasesDetectionJob (p. 130) operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StartSentimentDetectionJob

Starts an asynchronous sentiment detection job for a collection of documents. use the DescribeSentimentDetectionJob (p. 132) operation to track the status of a job.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "LanguageCode": "string",
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 188)**

A unique identifier for the request. If you don't set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 188)**

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data. For more information, see https://docs.aws.amazon.com/comprehend/latest/dg/access-control-managing-permissions.html#auth-role-permissions.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**InputDataConfig (p. 188)**

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 238) object

Required: Yes

**JobName (p. 188)**

The identifier of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**LanguageCode (p. 188)**

The language of the input documents. You can specify English ("en") or Spanish ("es"). All documents must be in the same language.

Type: String

Valid Values: `en | es`

Required: Yes

**OutputDataConfig (p. 188)**

Specifies where to send the output files.

Type: OutputDataConfig (p. 243) object

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 189)**

The identifier generated for the job. To get the status of a job, use this identifier with the DescribeSentimentDetectionJob (p. 132) operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 189)**

The status of the job.

- SUBMITTED - The job has been received and is queued for processing.

- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. To get details, use the DescribeSentimentDetectionJob (p. 132) operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StartTopicsDetectionJob

Starts an asynchronous topic detection job. Use the `DescribeTopicDetectionJob` operation to track the status of a job.

## Request Syntax

```
{
   "ClientRequestToken": "string",
   "DataAccessRoleArn": "string",
   "InputDataConfig": {
      "InputFormat": "string",
      "S3Uri": "string"
   },
   "JobName": "string",
   "NumberOfTopics": number,
   "OutputDataConfig": {
      "S3Uri": "string"
   }
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**ClientRequestToken (p. 191)**

A unique identifier for the request. If you do not set the client request token, Amazon Comprehend generates one.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: `^[a-zA-Z0-9-]+$`

Required: No

**DataAccessRoleArn (p. 191)**

The Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data. For more information, see https://docs.aws.amazon.com/comprehend/latest/dg/access-control-managing-permissions.html#auth-role-permissions.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: Yes

**InputDataConfig (p. 191)**

Specifies the format and location of the input data for the job.

Type: InputDataConfig (p. 238) object

Required: Yes

**JobName (p. 191)**

The identifier of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**NumberOfTopics (p. 191)**

The number of topics to detect.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**OutputDataConfig (p. 191)**

Specifies where to send the output files. The output is a compressed archive with two files, `topic-terms.csv` that lists the terms associated with each topic, and `doc-topics.csv` that lists the documents associated with each topic

Type: OutputDataConfig (p. 243) object

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 192)**

The identifier generated for the job. To get the status of the job, use this identifier with the `DescribeTopicDetectionJob` operation.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 192)**

The status of the job:

- SUBMITTED - The job has been received and is queued for processing.
- IN_PROGRESS - Amazon Comprehend is processing the job.
- COMPLETED - The job was successfully completed and the output is available.
- FAILED - The job did not complete. To get details, use the `DescribeTopicDetectionJob` operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**TooManyRequestsException**

The number of requests exceeds the limit. Resubmit your request later.

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

# StopDominantLanguageDetectionJob

Stops a dominant language detection job in progress.

If the job state is `IN_PROGRESS` the job is marked for termination and put into the `STOP_REQUESTED` state. If the job completes before it can be stopped, it is put into the `COMPLETED` state; otherwise the job is stopped and put into the `STOPPED` state.

If the job is in the `COMPLETED` or `FAILED` state when you call the `StopDominantLanguageDetectionJob` operation, the operation returns a 400 Internal Request Exception.

When a job is stopped, any documents already processed are written to the output location.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 194)**

The identifier of the dominant language detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 194)**

The identifier of the dominant language detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 194)**

Either `STOP_REQUESTED` if the job is currently running, or `STOPPED` if the job was previously stopped with the `StopDominantLanguageDetectionJob` operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try 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

# StopEntitiesDetectionJob

Stops an entities detection job in progress.

If the job state is `IN_PROGRESS` the job is marked for termination and put into the `STOP_REQUESTED` state. If the job completes before it can be stopped, it is put into the `COMPLETED` state; otherwise the job is stopped and put into the `STOPPED` state.

If the job is in the `COMPLETED` or `FAILED` state when you call the `StopDominantLanguageDetectionJob` operation, the operation returns a 400 Internal Request Exception.

When a job is stopped, any documents already processed are written to the output location.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 196)**

The identifier of the entities detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 196)**

The identifier of the entities detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 196)**

Either `STOP_REQUESTED` if the job is currently running, or `STOPPED` if the job was previously stopped with the `StopEntitiesDetectionJob` operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try 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

# StopKeyPhrasesDetectionJob

Stops a key phrases detection job in progress.

If the job state is `IN_PROGRESS` the job is marked for termination and put into the `STOP_REQUESTED` state. If the job completes before it can be stopped, it is put into the `COMPLETED` state; otherwise the job is stopped and put into the `STOPPED` state.

If the job is in the `COMPLETED` or `FAILED` state when you call the `StopDominantLanguageDetectionJob` operation, the operation returns a 400 Internal Request Exception.

When a job is stopped, any documents already processed are written to the output location.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 198)**

The identifier of the key phrases detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 198)**

The identifier of the key phrases detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 198)**

Either `STOP_REQUESTED` if the job is currently running, or `STOPPED` if the job was previously stopped with the `StopKeyPhrasesDetectionJob` operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try 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

# StopSentimentDetectionJob

Stops a sentiment detection job in progress.

If the job state is `IN_PROGRESS` the job is marked for termination and put into the `STOP_REQUESTED` state. If the job completes before it can be stopped, it is put into the `COMPLETED` state; otherwise the job is be stopped and put into the `STOPPED` state.

If the job is in the `COMPLETED` or `FAILED` state when you call the `StopDominantLanguageDetectionJob` operation, the operation returns a 400 Internal Request Exception.

When a job is stopped, any documents already processed are written to the output location.

## Request Syntax

```
{
   "JobId": "string"
}
```

## Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 254).

The request accepts the following data in JSON format.

**JobId (p. 200)**

The identifier of the sentiment detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: Yes

## Response Syntax

```
{
   "JobId": "string",
   "JobStatus": "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.

**JobId (p. 200)**

The identifier of the sentiment detection job to stop.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

**JobStatus (p. 200)**

Either `STOP_REQUESTED` if the job is currently running, or `STOPPED` if the job was previously stopped with the `StopSentimentDetectionJob` operation.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

## Errors

For information about the errors that are common to all actions, see Common Errors (p. 252).

**InternalServerException**

An internal server error occurred. Retry your request.

HTTP Status Code: 500

**InvalidRequestException**

The request is invalid.

HTTP Status Code: 400

**JobNotFoundException**

The specified job was not found. Check the job ID and try 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

# Data Types

The following data types are supported:

# BatchDetectDominantLanguageItemResult

The result of calling the BatchDetectDominantLanguage (p. 94) operation. The operation returns one object for each document that is successfully processed by the operation.

## Contents

**Index**

The zero-based index of the document in the input list.

Type: Integer

Required: No

**Languages**

One or more DominantLanguage (p. 218) objects describing the dominant languages in the document.

Type: Array of DominantLanguage (p. 218) 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

# BatchDetectEntitiesItemResult

The result of calling the BatchDetectKeyPhrases (p. 100) operation. The operation returns one object for each document that is successfully processed by the operation.

## Contents

**Entities**

One or more Entity (p. 225) objects, one for each entity detected in the document.

Type: Array of Entity (p. 225) objects

Required: No

**Index**

The zero-based index of the document in the input list.

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

# BatchDetectKeyPhrasesItemResult

The result of calling the BatchDetectKeyPhrases (p. 100) operation. The operation returns one object for each document that is successfully processed by the operation.

## Contents

**Index**

The zero-based index of the document in the input list.

Type: Integer

Required: No

**KeyPhrases**

One or more KeyPhrase (p. 239) objects, one for each key phrase detected in the document.

Type: Array of KeyPhrase (p. 239) 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

# BatchDetectSentimentItemResult

The result of calling the BatchDetectSentiment (p. 103) operation. The operation returns one object for each document that is successfully processed by the operation.

## Contents

**Index**

The zero-based index of the document in the input list.

Type: Integer

Required: No

**Sentiment**

The sentiment detected in the document.

Type: String

Valid Values: `POSITIVE | NEGATIVE | NEUTRAL | MIXED`

Required: No

**SentimentScore**

The level of confidence that Amazon Comprehend has in the accuracy of its sentiment detection.

Type: SentimentScore (p. 248) object

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

# BatchDetectSyntaxItemResult

The result of calling the BatchDetectSyntax (p. 106) operation. The operation returns one object that is successfully processed by the operation.

## Contents

**Index**

The zero-based index of the document in the input list.

Type: Integer

Required: No

**SyntaxTokens**

The syntax tokens for the words in the document, one token for each word.

Type: Array of SyntaxToken (p. 249) 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

# BatchItemError

Describes an error that occurred while processing a document in a batch. The operation returns on `BatchItemError` object for each document that contained an error.

## Contents

**ErrorCode**

The numeric error code of the error.

Type: String

Length Constraints: Minimum length of 1.

Required: No

**ErrorMessage**

A text description of the error.

Type: String

Length Constraints: Minimum length of 1.

Required: No

**Index**

The zero-based index of the document in the input list.

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

# ClassifierEvaluationMetrics

Describes the result metrics for the test data associated with an documentation classifier.

## Contents

**Accuracy**

The fraction of the labels that were correct recognized. It is computed by dividing the number of labels in the test documents that were correctly recognized by the total number of labels in the test documents.

Type: Double

Required: No

**F1Score**

A measure of how accurate the classifier results are for the test data. It is derived from the `Precision` and `Recall` values. The `F1Score` is the harmonic average of the two scores. The highest score is 1, and the worst score is 0.

Type: Double

Required: No

**Precision**

A measure of the usefulness of the classifier results in the test data. High precision means that the classifier returned substantially more relevant results than irrelevant ones.

Type: Double

Required: No

**Recall**

A measure of how complete the classifier results are for the test data. High recall means that the classifier returned most of the relevant results.

Type: Double

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

# ClassifierMetadata

Provides information about a document classifier.

## Contents

**EvaluationMetrics**

Describes the result metrics for the test data associated with an documentation classifier.

Type: ClassifierEvaluationMetrics (p. 209) object

Required: No

**NumberOfLabels**

The number of labels in the input data.

Type: Integer

Required: No

**NumberOfTestDocuments**

The number of documents in the input data that were used to test the classifier. Typically this is 10 to 20 percent of the input documents.

Type: Integer

Required: No

**NumberOfTrainedDocuments**

The number of documents in the input data that were used to train the classifier. Typically this is 80 to 90 percent of the input documents.

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

# DocumentClassificationJobFilter

Provides information for filtering a list of document classification jobs. For more information, see the ListDocumentClassificationJobs (p. 150) operation. You can provide only one filter parameter in each request.

## Contents

**JobName**

Filters on the name of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

Filters the list based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**SubmitTimeAfter**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted before the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted after the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

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

# DocumentClassificationJobProperties

Provides information about a document classification job.

## Contents

**DataAccessRoleArn**

The Amazon Resource Name (ARN) of the AWS identity and Access Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**DocumentClassifierArn**

The Amazon Resource Name (ARN) that identifies the document classifier.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:document-classifier/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**EndTime**

The time that the document classification job completed.

Type: Timestamp

Required: No

**InputDataConfig**

The input data configuration that you supplied when you created the document classification job.

Type: InputDataConfig (p. 238) object

Required: No

**JobId**

The identifier assigned to the document classification job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobName**

The name that you assigned to the document classification job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

The current status of the document classification job. If the status is `FAILED`, the `Message` field shows the reason for the failure.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**Message**

A description of the status of the job.

Type: String

Required: No

**OutputDataConfig**

The output data configuration that you supplied when you created the document classification job.

Type: OutputDataConfig (p. 243) object

Required: No

**SubmitTime**

The time that the document classification job was submitted for processing.

Type: Timestamp

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

# DocumentClassifierFilter

Provides information for filtering a list of document classifiers. You can only specify one filtering parameter in a request. For more information, see the ListDocumentClassifiers (p. 153) operation.

## Contents

**Status**

Filters the list of classifiers based on status.

Type: String

Valid Values: `SUBMITTED | TRAINING | DELETING | IN_ERROR | TRAINED`

Required: No

**SubmitTimeAfter**

Filters the list of classifiers based on the time that the classifier was submitted for processing. Returns only classifiers submitted after the specified time. Classifiers are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of classifiers based on the time that the classifier was submitted for processing. Returns only classifiers submitted before the specified time. Classifiers are returned in ascending order, oldest to newest.

Type: Timestamp

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

# DocumentClassifierInputDataConfig

The input properties for training a document classifier.

For more information on how the input file is formatted, see Creating Training Data (p. 67).

## Contents

**S3Uri**

The Amazon S3 URI for the input data. The S3 bucket must be in the same region as the API endpoint that you are calling. The URI can point to a single input file or it can provide the prefix for a collection of input files.

For example, if you use the URI `S3://bucketName/prefix`, if the prefix is a single file, Amazon Comprehend uses that file as input. If more than one file begins with the prefix, Amazon Comprehend uses all of them as input.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9](/.*)?`

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

# DocumentClassifierProperties

Provides information about a document classifier.

## Contents

**ClassifierMetadata**

Information about the document classifier, including the number of documents used for training the classifier, the number of documents used for test the classifier, and an accuracy rating.

Type: ClassifierMetadata (p. 210) object

Required: No

**DataAccessRoleArn**

The Amazon Resource Name (ARN) of the AWS Identity and Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**DocumentClassifierArn**

The Amazon Resource Name (ARN) that identifies the document classifier.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:document-classifier/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**EndTime**

The time that training the document classifier completed.

Type: Timestamp

Required: No

**InputDataConfig**

The input data configuration that you supplied when you created the document classifier for training.

Type: DocumentClassifierInputDataConfig (p. 215) object

Required: No

**LanguageCode**

The language code for the language of the documents that the classifier was trained on.

Type: String

Valid Values: `en | es`

Required: No

**Message**

Additional information about the status of the classifier.

Type: String

Required: No

**Status**

The status of the document classifier. The the status is `TRAINED` the classifier is ready to use. If the status is `FAILED` you can see additional information about why the classifier wasn't trained in the `Message` field.

Type: String

Valid Values: `SUBMITTED | TRAINING | DELETING | IN_ERROR | TRAINED`

Required: No

**SubmitTime**

The time that the document classifier was submitted for training.

Type: Timestamp

Required: No

**TrainingEndTime**

The time that training of the document classifier was completed. Indicates the time when the training completes on documentation classifiers. You are billed for the time interval between this time and the value of TrainingStartTime.

Type: Timestamp

Required: No

**TrainingStartTime**

Indicates the time when the training starts on documentation classifiers. You are billed for the time interval between this time and the value of TrainingEndTime.

Type: Timestamp

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

# DominantLanguage

Returns the code for the dominant language in the input text and the level of confidence that Amazon Comprehend has in the accuracy of the detection.

## Contents

**LanguageCode**

The RFC 5646 language code for the dominant language. For more information about RFC 5646, see Tags for Identifying Languages on the *IETF Tools* web site.

Type: String

Length Constraints: Minimum length of 1.

Required: No

**Score**

The level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Float

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

# DominantLanguageDetectionJobFilter

Provides information for filtering a list of dominant language detection jobs. For more information, see the ListDominantLanguageDetectionJobs (p. 156) operation.

## Contents

**JobName**

Filters on the name of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

Filters the list of jobs based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**SubmitTimeAfter**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted after the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted before the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

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

# DominantLanguageDetectionJobProperties

Provides information about a dominant language detection job.

## Contents

**DataAccessRoleArn**

The Amazon Resource Name (ARN) that gives Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**EndTime**

The time that the dominant language detection job completed.

Type: Timestamp

Required: No

**InputDataConfig**

The input data configuration that you supplied when you created the dominant language detection job.

Type: InputDataConfig (p. 238) object

Required: No

**JobId**

The identifier assigned to the dominant language detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobName**

The name that you assigned to the dominant language detection job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

The current status of the dominant language detection job. If the status is `FAILED`, the `Message` field shows the reason for the failure.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**Message**

A description for the status of a job.

Type: String

Required: No

**OutputDataConfig**

The output data configuration that you supplied when you created the dominant language detection job.

Type: OutputDataConfig (p. 243) object

Required: No

**SubmitTime**

The time that the dominant language detection job was submitted for processing.

Type: Timestamp

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

# EntitiesDetectionJobFilter

Provides information for filtering a list of dominant language detection jobs. For more information, see the ListEntitiesDetectionJobs (p. 159) operation.

## Contents

**JobName**

Filters on the name of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

Filters the list of jobs based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**SubmitTimeAfter**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted after the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted before the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

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

# EntitiesDetectionJobProperties

Provides information about an entities detection job.

## Contents

**DataAccessRoleArn**

The Amazon Resource Name (ARN) that gives Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**EndTime**

The time that the entities detection job completed

Type: Timestamp

Required: No

**EntityRecognizerArn**

The Amazon Resource Name (ARN) that identifies the entity recognizer.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:entity-recognizer/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**InputDataConfig**

The input data configuration that you supplied when you created the entities detection job.

Type: InputDataConfig (p. 238) object

Required: No

**JobId**

The identifier assigned to the entities detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobName**

The name that you assigned the entities detection job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

The current status of the entities detection job. If the status is `FAILED`, the `Message` field shows the reason for the failure.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**LanguageCode**

The language code of the input documents.

Type: String

Valid Values: `en | es`

Required: No

**Message**

A description of the status of a job.

Type: String

Required: No

**OutputDataConfig**

The output data configuration that you supplied when you created the entities detection job.

Type: OutputDataConfig (p. 243) object

Required: No

**SubmitTime**

The time that the entities detection job was submitted for processing.

Type: Timestamp

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

# Entity

Provides information about an entity.

## Contents

**BeginOffset**

A character offset in the input text that shows where the entity begins (the first character is at position 0). The offset returns the position of each UTF-8 code point in the string. A *code point* is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

**EndOffset**

A character offset in the input text that shows where the entity ends. The offset returns the position of each UTF-8 code point in the string. A *code point* is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

**Score**

The level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Float

Required: No

**Text**

The text of the entity.

Type: String

Length Constraints: Minimum length of 1.

Required: No

**Type**

The entity's type.

Type: String

Valid Values: `PERSON | LOCATION | ORGANIZATION | COMMERCIAL_ITEM | EVENT | DATE | QUANTITY | TITLE | OTHER`

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

# EntityRecognizerAnnotations

Describes the annotations associated with a entity recognizer.

## Contents

**S3Uri**

Specifies the Amazon S3 location where the annotations for an entity recognizer are located. The URI must be in the same region as the API endpoint that you are calling.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9](/.*)?`

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

# EntityRecognizerDocuments

Describes the training documents submitted with an entity recognizer.

## Contents

**S3Uri**

Specifies the Amazon S3 location where the training documents for an entity recognizer are located. The URI must be in the same region as the API endpoint that you are calling.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9](/.*)?`

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

# EntityRecognizerEntityList

Describes the entity recognizer submitted with an entity recognizer.

## Contents

**S3Uri**

Specifies the Amazon S3 location where the entity list is located. The URI must be in the same region as the API endpoint that you are calling.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9](/.*)?`

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

# EntityRecognizerEvaluationMetrics

Detailed information about the accuracy of an entity recognizer.

## Contents

**F1Score**

A measure of how accurate the recognizer results are for the test data. It is derived from the `Precision` and `Recall` values. The `F1Score` is the harmonic average of the two scores. The highest score is 1, and the worst score is 0.

Type: Double

Required: No

**Precision**

A measure of the usefulness of the recognizer results in the test data. High precision means that the recognizer returned substantially more relevant results than irrelevant ones.

Type: Double

Required: No

**Recall**

A measure of how complete the recognizer results are for the test data. High recall means that the recognizer returned most of the relevant results.

Type: Double

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

# EntityRecognizerFilter

Provides information for filtering a list of entity recognizers. You can only specify one filtering parameter in a request. For more information, see the ListEntityRecognizers (p. 162) operation./>

## Contents

**Status**

The status of an entity recognizer.

Type: String

Valid Values: `SUBMITTED | TRAINING | DELETING | IN_ERROR | TRAINED`

Required: No

**SubmitTimeAfter**

Filters the list of entities based on the time that the list was submitted for processing. Returns only jobs submitted after the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of entities based on the time that the list was submitted for processing. Returns only jobs submitted before the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

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

# EntityRecognizerInputDataConfig

Specifies the format and location of the input data.

## Contents

**Annotations**

S3 location of the annotations file for an entity recognizer.

Type: EntityRecognizerAnnotations (p. 227) object

Required: No

**Documents**

S3 location of the documents folder for an entity recognizer

Type: EntityRecognizerDocuments (p. 228) object

Required: Yes

**EntityList**

S3 location of the entity list for an entity recognizer.

Type: EntityRecognizerEntityList (p. 229) object

Required: No

**EntityTypes**

The entity types in the input data for an entity recognizer.

Type: Array of EntityTypesListItem (p. 237) objects

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

# EntityRecognizerMetadata

Detailed information about an entity recognizer.

## Contents

**EntityTypes**

Entity types from the metadata of an entity recognizer.

Type: Array of EntityRecognizerMetadataEntityTypesListItem (p. 234) objects

Required: No

**EvaluationMetrics**

Detailed information about the accuracy of an entity recognizer.

Type: EntityRecognizerEvaluationMetrics (p. 230) object

Required: No

**NumberOfTestDocuments**

The number of documents in the input data that were used to test the entity recognizer. Typically this is 10 to 20 percent of the input documents.

Type: Integer

Required: No

**NumberOfTrainedDocuments**

The number of documents in the input data that were used to train the entity recognizer. Typically this is 80 to 90 percent of the input documents.

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

# EntityRecognizerMetadataEntityTypesListItem

Individual item from the list of entity types in the metadata of an entity recognizer.

## Contents

**Type**

Type of entity from the list of entity types in the metadata of an entity recognizer.

Type: String

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

# EntityRecognizerProperties

Describes information about an entity recognizer.

## Contents

**DataAccessRoleArn**

The Amazon Resource Name (ARN) of the AWS Identity and Management (IAM) role that grants Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**EndTime**

The time that the recognizer creation completed.

Type: Timestamp

Required: No

**EntityRecognizerArn**

The Amazon Resource Name (ARN) that identifies the entity recognizer.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:comprehend:[a-zA-Z0-9-]*:[0-9]{12}:entity-recognizer/[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: No

**InputDataConfig**

The input data properties of an entity recognizer.

Type: EntityRecognizerInputDataConfig (p. 232) object

Required: No

**LanguageCode**

The language of the input documents. All documents must be in the same language. Only English ("en") is currently supported.

Type: String

Valid Values: `en | es`

Required: No

**Message**

A description of the status of the recognizer.

Type: String

Required: No

**RecognizerMetadata**

Provides information about an entity recognizer.

Type: EntityRecognizerMetadata (p. 233) object

Required: No

**Status**

Provides the status of the entity recognizer.

Type: String

Valid Values: `SUBMITTED | TRAINING | DELETING | IN_ERROR | TRAINED`

Required: No

**SubmitTime**

The time that the recognizer was submitted for processing.

Type: Timestamp

Required: No

**TrainingEndTime**

The time that training of the entity recognizer was completed.

Type: Timestamp

Required: No

**TrainingStartTime**

The time that training of the entity recognizer started.

Type: Timestamp

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

# EntityTypesListItem

Information about an individual item on a list of entity types.

## Contents

**Type**

Entity type of an item on an entity type list.

Type: String

Length Constraints: Maximum length of 64.

Pattern: `[_A-Z0-9]+`

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

# InputDataConfig

The input properties for a topic detection job.

## Contents

**InputFormat**

Specifies how the text in an input file should be processed:

- `ONE_DOC_PER_FILE` - Each file is considered a separate document. Use this option when you are processing large documents, such as newspaper articles or scientific papers.
- `ONE_DOC_PER_LINE` - Each line in a file is considered a separate document. Use this option when you are processing many short documents, such as text messages.

Type: String

Valid Values: `ONE_DOC_PER_FILE | ONE_DOC_PER_LINE`

Required: No

**S3Uri**

The Amazon S3 URI for the input data. The URI must be in same region as the API endpoint that you are calling. The URI can point to a single input file or it can provide the prefix for a collection of data files.

For example, if you use the URI `S3://bucketName/prefix`, if the prefix is a single file, Amazon Comprehend uses that file as input. If more than one file begins with the prefix, Amazon Comprehend uses all of them as input.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9](/.*)?`

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

# KeyPhrase

Describes a key noun phrase.

## Contents

**BeginOffset**

A character offset in the input text that shows where the key phrase begins (the first character is at position 0). The offset returns the position of each UTF-8 code point in the string. A *code point* is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

**EndOffset**

A character offset in the input text where the key phrase ends. The offset returns the position of each UTF-8 code point in the string. A `code point` is the abstract character from a particular graphical representation. For example, a multi-byte UTF-8 character maps to a single code point.

Type: Integer

Required: No

**Score**

The level of confidence that Amazon Comprehend has in the accuracy of the detection.

Type: Float

Required: No

**Text**

The text of a key noun phrase.

Type: String

Length Constraints: Minimum length of 1.

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

# KeyPhrasesDetectionJobFilter

Provides information for filtering a list of dominant language detection jobs. For more information, see the ListKeyPhrasesDetectionJobs (p. 165) operation.

## Contents

**JobName**

Filters on the name of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

Filters the list of jobs based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**SubmitTimeAfter**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted after the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted before the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

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

# KeyPhrasesDetectionJobProperties

Provides information about a key phrases detection job.

## Contents

**DataAccessRoleArn**

The Amazon Resource Name (ARN) that gives Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**EndTime**

The time that the key phrases detection job completed.

Type: Timestamp

Required: No

**InputDataConfig**

The input data configuration that you supplied when you created the key phrases detection job.

Type: InputDataConfig (p. 238) object

Required: No

**JobId**

The identifier assigned to the key phrases detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobName**

The name that you assigned the key phrases detection job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

The current status of the key phrases detection job. If the status is `FAILED`, the `Message` field shows the reason for the failure.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**LanguageCode**

The language code of the input documents.

Type: String

Valid Values: `en | es`

Required: No

**Message**

A description of the status of a job.

Type: String

Required: No

**OutputDataConfig**

The output data configuration that you supplied when you created the key phrases detection job.

Type: OutputDataConfig (p. 243) object

Required: No

**SubmitTime**

The time that the key phrases detection job was submitted for processing.

Type: Timestamp

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

# OutputDataConfig

Provides configuration parameters for the output of topic detection jobs.

## Contents

**S3Uri**

When you use the `OutputDataConfig` object with asynchronous operations, you specify the Amazon S3 location where you want to write the output data. The URI must be in the same region as the API endpoint that you are calling. The location is used as the prefix for the actual location of the output file.

When the topic detection job is finished, the service creates an output file in a directory specific to the job. The `S3Uri` field contains the location of the output file, called `output.tar.gz`. It is a compressed archive that contains the ouput of the operation.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `s3://[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9](/.*)?`

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

# PartOfSpeechTag

Identifies the part of speech represented by the token and gives the confidence that Amazon Comprehend has that the part of speech was correctly identified. For more information about the parts of speech that Amazon Comprehend can identify, see Analyze Syntax (p. 53).

## Contents

**Score**

The confidence that Amazon Comprehend has that the part of speech was correctly identified.

Type: Float

Required: No

**Tag**

Identifies the part of speech that the token represents.

Type: String

Valid Values: `ADJ | ADP | ADV | AUX | CONJ | DET | INTJ | NOUN | NUM | O | PART | PRON | PROPN | PUNCT | SCONJ | SYM | VERB`

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

# SentimentDetectionJobFilter

Provides information for filtering a list of dominant language detection jobs. For more information, see the ListSentimentDetectionJobs (p. 168) operation.

## Contents

**JobName**

Filters on the name of the job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

Filters the list of jobs based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**SubmitTimeAfter**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted after the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of jobs based on the time that the job was submitted for processing. Returns only jobs submitted before the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

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

# SentimentDetectionJobProperties

Provides information about a sentiment detection job.

## Contents

**DataAccessRoleArn**

The Amazon Resource Name (ARN) that gives Amazon Comprehend read access to your input data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 2048.

Pattern: `arn:aws(-[^:]+)?:iam::[0-9]{12}:role/.+`

Required: No

**EndTime**

The time that the sentiment detection job ended.

Type: Timestamp

Required: No

**InputDataConfig**

The input data configuration that you supplied when you created the sentiment detection job.

Type: InputDataConfig (p. 238) object

Required: No

**JobId**

The identifier assigned to the sentiment detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobName**

The name that you assigned to the sentiment detection job

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

The current status of the sentiment detection job. If the status is `FAILED`, the `Messages` field shows the reason for the failure.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**LanguageCode**

The language code of the input documents.

Type: String

Valid Values: `en | es`

Required: No

**Message**

A description of the status of a job.

Type: String

Required: No

**OutputDataConfig**

The output data configuration that you supplied when you created the sentiment detection job.

Type: OutputDataConfig (p. 243) object

Required: No

**SubmitTime**

The time that the sentiment detection job was submitted for processing.

Type: Timestamp

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

# SentimentScore

Describes the level of confidence that Amazon Comprehend has in the accuracy of its detection of sentiments.

## Contents

**Mixed**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `MIXED` sentiment.

Type: Float

Required: No

**Negative**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `NEGATIVE` sentiment.

Type: Float

Required: No

**Neutral**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `NEUTRAL` sentiment.

Type: Float

Required: No

**Positive**

The level of confidence that Amazon Comprehend has in the accuracy of its detection of the `POSITIVE` sentiment.

Type: Float

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

# SyntaxToken

Represents a work in the input text that was recognized and assigned a part of speech. There is one syntax token record for each word in the source text.

## Contents

**BeginOffset**

The zero-based offset from the beginning of the source text to the first character in the word.

Type: Integer

Required: No

**EndOffset**

The zero-based offset from the beginning of the source text to the last character in the word.

Type: Integer

Required: No

**PartOfSpeech**

Provides the part of speech label and the confidence level that Amazon Comprehend has that the part of speech was correctly identified. For more information, see Analyze Syntax (p. 53).

Type: PartOfSpeechTag (p. 244) object

Required: No

**Text**

The word that was recognized in the source text.

Type: String

Length Constraints: Minimum length of 1.

Required: No

**TokenId**

A unique identifier for a token.

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

# TopicsDetectionJobFilter

Provides information for filtering topic detection jobs. For more information, see ListTopicsDetectionJobs (p. 171).

## Contents

**JobName**

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

Filters the list of topic detection jobs based on job status. Returns only jobs with the specified status.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**SubmitTimeAfter**

Filters the list of jobs based on the time that the job was submitted for processing. Only returns jobs submitted after the specified time. Jobs are returned in ascending order, oldest to newest.

Type: Timestamp

Required: No

**SubmitTimeBefore**

Filters the list of jobs based on the time that the job was submitted for processing. Only returns jobs submitted before the specified time. Jobs are returned in descending order, newest to oldest.

Type: Timestamp

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

# TopicsDetectionJobProperties

Provides information about a topic detection job.

## Contents

**EndTime**

The time that the topic detection job was completed.

Type: Timestamp

Required: No

**InputDataConfig**

The input data configuration supplied when you created the topic detection job.

Type: InputDataConfig (p. 238) object

Required: No

**JobId**

The identifier assigned to the topic detection job.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobName**

The name of the topic detection job.

Type: String

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

Pattern: `^([\p{L}\p{Z}\p{N}_.:/=+\-%@]*)$`

Required: No

**JobStatus**

The current status of the topic detection job. If the status is `Failed`, the reason for the failure is shown in the `Message` field.

Type: String

Valid Values: `SUBMITTED | IN_PROGRESS | COMPLETED | FAILED | STOP_REQUESTED | STOPPED`

Required: No

**Message**

A description for the status of a job.

Type: String

Required: No

**NumberOfTopics**

The number of topics to detect supplied when you created the topic detection job. The default is 10.

Type: Integer

Required: No

**OutputDataConfig**

The output data configuration supplied when you created the topic detection job.

Type: OutputDataConfig (p. 243) object

Required: No

**SubmitTime**

The time that the topic detection job was submitted for processing.

Type: Timestamp

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

# Document History for Amazon Comprehend

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

- **Latest documentation update:** October 10, 2018

| update-history-change | update-history-description | update-history-date |
|---|---|---|
| Region expansion | Amazon Comprehend is now available in EU (Frankfurt) (eu-central-1). | October 10, 2018 |
| Language expansion | In addition to English and Spanish Amazon Comprehend can now also examine documents in French, German, Italian, and Portuguese. For more information, see Supported Languages in Amazon Comprehend. | October 10, 2018 |
| Region expansion | Amazon Comprehend is now available in Asia Pacific (Sydney) (ap-southeast-2). | August 15, 2018 |
| New feature | Amazon Comprehend now parses documents to discover the syntax of a document and the part of speech for each word. For more information, see Syntax. | July 17, 2018 |
| New feature | Amazon Comprehend now supports asynchronous batch processing for language, key phrase, entity, and sentiment detection. For more information, see Asynchronous Batch Processing . | June 27, 2018 |
| New guide (p. 256) | This is the first release of the *Amazon Comprehend Developer Guide*. | November 29, 2017 |