



Developer Guide

Amazon Polly



Amazon Polly: Developer Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the [AWS European Sovereign Cloud User Guide](#).

Table of Contents

What Is Amazon Polly?	1
How it works	1
Benefits	2
Are you a first-time user?	3
Working with AWS SDKs	3
Getting started	4
Signing up for AWS	4
Sign up for an AWS account	5
Setting up the AWS CLI	5
Reconfiguring the AWS CLI	6
Synthesizing speech example	8
Bidirectional streaming	12
SynthesizeSpeech and StartSpeechSynthesisStream compared	12
Sending text and receiving audio	14
Open the stream	14
Send text	14
Receive audio	15
Close the stream	16
Code examples	16
Voices in Amazon Polly	26
Available voices	26
Brand voices	34
Bilingual voices	34
Accented bilingual voices	35
Fully bilingual voices	35
Applying the newscaster voice	36
Listening to voices	39
Timing a voice speed	39
Changing a voice speed	40
Languages in Amazon Polly	42
Arabic (arb)	45
Arabic (Gulf) (ar-AE)	50
Catalan (ca-ES)	56
Chinese (Cantonese) (yue-CN)	60

Chinese (Mandarin) (cmn-CN)	64
Czech (cs-CZ)	69
Danish (da-DK)	73
Dutch (Belgian) (nl-BE)	77
Dutch (nl-NL)	81
English (US) (en-US)	85
English (Australian) (en-AU)	88
English (British) (en-GB)	92
English (Indian) (en-IN)	97
English (Ireland) (en-IE)	101
English (New Zealand) (en-NZ)	104
English (Singaporean) (en-SG)	110
English (South African) (en-ZA)	114
English (Welsh) (en-GB-WLS)	119
Finnish (fi-FI)	123
French (fr-FR)	128
French (Belgian) (fr-BE)	131
French (Canadian) (fr-CA)	135
German (de-DE)	138
German (Austrian) (de-AT)	142
German (Swiss standard) (de-CH)	147
Hindi (hi-IN)	151
Icelandic (is-IS)	155
Italian (it-IT)	159
Japanese (ja-JP)	162
Korean (ko-KR)	166
Norwegian (nb-NO)	169
Polish (pl-PL)	173
Portuguese (pt-PT)	176
Portuguese (Brazilian) (pt-BR)	179
Romanian (ro-RO)	182
Russian (ru-RU)	185
Spanish (es-ES)	189
Spanish (Mexican) (es-MX)	192
Spanish (US) (es-US)	195
Swedish (sv-SE)	198

Turkish (tr-TR)	202
Welsh (cy-GB)	206
Voice engines	211
Generative engine	211
Available generative voices	212
Feature and region compatibility	214
Long-form engine	216
Available long-form voices	217
Feature and region compatibility	217
Neural engine	218
Available neural voices	219
Feature and region compatibility	223
Standard engine	225
Available Standard voices	225
Feature and region compatibility	228
Choosing a voice engine	230
Speech marks	231
Speech mark types	231
Visemes and Amazon Polly	232
Speech mark output	233
Requesting speech marks	234
Speech marks without SSML example	236
Speech marks with SSML example	237
Using SSML	239
Reserved characters	240
Using SSML on the console	242
Using SSML with the Synthesize-Speech command	244
Synthesizing an SSML-enhanced document	245
Supported SSML tags	246
Identifying SSML-enhanced text	248
Adding a pause	249
Emphasizing words	250
Specifying another language for specific words	251
Placing a custom tag in your text	252
Adding a pause between paragraphs	253
Using phonetic pronunciation	253

Controlling volume, speaking rate, and pitch	255
Setting a maximum duration for synthesized speech	257
Adding a pause between sentences	261
Controlling how special types of words are spoken	262
Pronouncing acronyms and abbreviations	265
Improving pronunciation by specifying parts of speech	266
Adding the sound of breathing	267
Newscaster speaking style	271
Adding dynamic range compression	272
Speaking softly	274
Controlling timbre	275
Whispering	276
Managing lexicons	278
Using multiple lexicons	279
Uploading a lexicon	280
Applying lexicons (Synthesizing Speech)	286
Filtering the lexicon list on the console	289
Downloading lexicons on the console	291
Deleting a lexicon	292
Long audio files	294
Setting up the IAM policy for asynchronous synthesis	295
Creating long audio files	295
Quotas	300
Supported regions	301
Quotas and throttle rates	301
Concurrent requests	302
Best practices to mitigate throttling	302
Pronunciation lexicons	303
SynthesizeSpeech API operations	303
SpeechSynthesisTask API operations	304
Speech Synthesis Markup Language (SSML)	304
Sample code and applications	305
Java samples	305
DeleteLexicon	306
DescribeVoices	308
GetLexicon	309

ListLexicons	311
PutLexicon	312
StartSpeechSynthesisTask	315
Speech Marks	318
SynthesizeSpeech	321
Python samples	323
DeleteLexicon	323
GetLexicon	324
ListLexicon	326
PutLexicon	326
StartSpeechSynthesisTask	328
SynthesizeSpeech	328
Java example	329
Python example	334
Python example: index.html	335
Python example: server.py	340
iOS example	347
Android example	349
Code examples	352
Basics	353
Actions	353
Scenarios	404
Convert text to speech and back to text	404
Create a lip-sync application	405
Create an application to analyze customer feedback	406
Getting started with text-to-speech synthesis	412
Security	420
Data Protection	421
Encryption at Rest	421
Encryption in Transit	422
Internetwork Traffic Privacy	422
Identity and Access Management	422
Audience	422
Authenticating with identities	423
Managing access using policies	424
How Amazon Polly works with IAM	426

Identity-based policy examples	432
Amazon Polly API Permissions Reference	438
Troubleshooting	440
Logging and Monitoring	442
Compliance Validation	442
Resilience	443
Infrastructure Security	443
Security Best Practices	443
Using Interface VPC Endpoints	444
Availability	444
Creating a VPC endpoint for Amazon Polly	445
Testing the connection between your VPC and Amazon Polly	445
Controlling access to your Amazon Polly endpoint	445
Support for VPC context keys	446
Logging Amazon Polly API calls with AWS CloudTrail	447
Amazon Polly information in CloudTrail	447
Example: Amazon Polly Log File Entries	448
CloudWatch integration	450
Getting CloudWatch Metrics (Console)	450
Getting CloudWatch metrics on the AWS CLI	450
Amazon Polly Metrics	451
Dimensions for Amazon Polly Metrics	453
API Reference	454
Document History	455

What Is Amazon Polly?

Amazon Polly is a cloud service that converts text into lifelike speech. You can use Amazon Polly to develop applications that increase engagement and accessibility. Amazon Polly supports multiple languages and includes a variety of lifelike voices. With Amazon Polly, you can build speech-enabled applications that work in multiple locations and use the ideal voice for your customers. Also, you only pay for the text you synthesize. You can also cache and replay Amazon Polly's generated speech at no additional cost.

Amazon Polly offers many voice options, including generative, long-form, neural, and standard text-to-speech (TTS) options. These voices deliver ground-breaking improvements in speech quality using new machine learning technology to offer the most natural and human-like text-to-speech voices possible. Neural TTS technology also supports a Newscaster speaking style, tailored to news narration use cases.

Common use cases for Amazon Polly include, but are not limited to: mobile applications such as newsreaders, games, eLearning platforms, accessibility applications for visually impaired people, and the rapidly growing segment of Internet of Things (IoT).

Amazon Polly is certified for use with regulated workloads for HIPAA (the Health Insurance Portability and Accountability Act of 1996), and Payment Card Industry Data Security Standard (PCI DSS).

How Amazon Polly works

Amazon Polly converts input text into life-like speech. To use an Amazon Polly voice, choose a [voice engine](#), call a speech synthesis method, provide the text that you want to synthesize, then specify an audio output format. Amazon Polly then synthesizes the provided text into a high-quality speech audio stream.

- **Input text** – Provide the text that you want to synthesize, and Amazon Polly returns an audio stream. You can provide the input as plaintext or in Speech Synthesis Markup Language (SSML) format. With SSML you can control various aspects of speech, such as pronunciation, volume, pitch, and speech rate. For more information, see [Generating speech from SSML documents](#).
- **Available voices** – Amazon Polly provides a portfolio of languages and a variety of voices, including a bilingual voice (for both English and Hindi). For most languages you can choose from several voices, both male and female. When launching a speech synthesis task, you specify the

voice ID, and then Amazon Polly uses this voice to convert the text to speech. Amazon Polly is not a translation service—the synthesized speech is in the same language as the text. Numbers represented as digits (for example, *53*, not *fifty-three*) are synthesized in the language of the voice and not the text. For more information, see [Voices in Amazon Polly](#).

- **Output format** – Amazon Polly can deliver the synthesized speech in multiple formats. You can select the audio format that suits your needs. For example, you might request the speech in the MP3 or Ogg Vorbis format for consumption by web and mobile applications. Or, you might request the PCM output format for consumption by AWS IoT devices and telephony solutions. For telephony applications, you can use Mu-law or A-law formats.

Note

To hear example Amazon Polly voices in your browser, see the [Amazon Polly product overview](#).

Benefits

Some of the benefits of using Amazon Polly include:

- **High quality** – Amazon Polly offers highly-performant generative, long-form, neural, and high-quality text-to-speech (TTS) voices. These technologies synthesize natural speech with high pronunciation accuracy (including abbreviations, acronym expansions, date/time interpretations, and homograph disambiguation).
- **Low latency** – Amazon Polly achieves fast responses, which makes it a viable option for low-latency use cases such as dialogue systems.
- **Support for a large portfolio of languages and voices** – Amazon Polly supports dozens of voices and languages, offering male and female voice options for most languages. This number will continue to increase as we bring more neural voices online. US English voices Matthew and Joanna can also use the Neural Newscaster speaking style, similar to what you might hear from a professional news anchor.
- **Cost-effective** – Amazon Polly's pay-per-use model means that there are no setup costs. Start small and scale up as your application grows.
- **Cloud-based solution** – On-device TTS solutions require significant computing resources, notably CPU power, RAM, and disk space. These can result in higher development costs and

higher power consumption on devices such as tablets, smartphones, and so on. In contrast, TTS conversion done in the AWS Cloud dramatically reduces local resource requirements. This enables support of all the available languages and voices with outstanding quality. Moreover, speech improvements are instantly available to all end users and don't require additional updates for devices.

 Note

To hear example Amazon Polly voices in your browser, see the [Amazon Polly product overview](#).

Are you a first-time user?

If you're a first-time user of Amazon Polly, we recommend that you read the following sections in the listed order:

1. [How Amazon Polly works](#) – This section introduces various Amazon Polly inputs and options that you can work with in order to create a simple experience.
2. [Getting started with Amazon Polly](#) – In this section, you set up your account and test Amazon Polly speech synthesis.
3. [Sample code and applications for Amazon Polly](#) – This section provides additional examples that you can use to explore Amazon Polly.

Using Amazon Polly with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

Getting started with Amazon Polly

Amazon Polly provides several API operations that you can easily integrate with your existing applications. For a list of supported operations, see [Actions](#).

You can perform almost all of the same operations on the Amazon Polly console and the AWS CLI. However, you can't listen to synthesized speech on the AWS CLI. To work with audio on the AWS CLI, save your text to a file. Then open the file in an audio application of your choice.

You can use either of the following options:

- **AWS SDKs** – When using the SDKs, your requests to Amazon Polly are automatically signed and authenticated using the credentials you provide. This is the recommended choice for building your applications.
- **AWS CLI** – You can use the AWS CLI to use Amazon Polly without writing any code.

Before you use Amazon Polly for the first time, you must sign up for AWS. When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Amazon Polly. You're charged only for the services and resources that you use. If you're a new AWS customer, you can get started with Amazon Polly with no charge. For more information, see [AWS Free Usage Tier](#).

The following sections describe how to get started using Amazon Polly.

Topics

- [Signing up for AWS](#)
- [Setting up the AWS CLI](#)
- [Reconfiguring the AWS CLI](#)

Signing up for AWS

Before you can use any AWS service, including Amazon Polly, you must sign up for AWS.

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see [Getting started with an AWS account](#) in the *AWS Account Management Reference Guide*.

For more information about IAM, see the following:

- [AWS Identity and Access Management \(IAM\)](#)
- [Getting started with IAM](#)
- [IAM User Guide](#)

Note

Note your AWS account ID. You will need it in the next steps.

Setting up the AWS CLI

Follow these steps to download and configure the AWS CLI to work with Amazon Polly.

To set up the AWS Command Line Interface

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. Add a named profile for the administrator user in the AWS CLI AWS Config file. You can use this profile when running the AWS CLI commands. For more information about named profiles, see [Named Profiles](#) in the *AWS Command Line Interface User Guide*.

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

For a list of available AWS Regions and those supported by Amazon Polly, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Note

If you're using a Region supported by Amazon Polly that you specified when you configured the AWS CLI, omit the following line from the AWS CLI code examples.

```
--region aws-region
```

3. Verify the setup by typing the following help command at the command prompt.

```
aws help
```

A list of valid AWS commands should appear in the AWS CLI window.

Reconfiguring the AWS CLI

If you've previously downloaded and configured the AWS CLI, Amazon Polly might be unavailable unless you reconfigure the AWS CLI. The following procedure checks to see if this is necessary.

To reactivate Amazon Polly from the AWS CLI

1. Verify the availability of Amazon Polly by typing the following help command at the AWS CLI command prompt.

```
aws polly help
```

If you see a description of Amazon Polly and a list of valid commands appears in the AWS CLI window, you can use Amazon Polly from the AWS CLI immediately. In this case, you can skip the rest of this procedure. If this is not displayed, continue with Step 2.

2. Activate Amazon Polly using one of the two following options:
 - a. Uninstall and reinstall the AWS CLI.

For instructions, see [Installing the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

or

- b. Download the file [service-2.json](#).

At the command prompt, run the following command.

```
aws configure add-model --service-model file://service-2.json --service-name  
polly
```

3. Reverify the availability of Amazon Polly.

```
aws polly help
```

The description of Amazon Polly should be visible.

Synthesizing speech with Amazon Polly example

This page presents a short speech synthesis example performed in the console, the AWS CLI, and with Python. This example performs speech synthesis from plain text, not SSML.

Console

Synthesize speech on the console

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Text-to-Speech** tab. The text field will load with example text so you can quickly try out Amazon Polly.
3. Turn off **SSML**.
4. Type or paste this text into the input box.

```
He was caught up in the game. In the middle of the 10/3/2014 W3C meeting he
shouted, "Score!" quite loudly.
```

5. Under **Engine**, choose **Generative, Long Form, Neural, or Standard**.
6. Choose a language and AWS Region, then choose a voice. (If you select **Neural** for **Engine**, only the languages and voices that support NTTS are available. All Standard and Long Form voices are disabled.)
7. To listen to the speech immediately, choose **Listen**.
8. To save the speech to a file, do one of the following:
 - a. Choose **Download**.
 - b. To change to a different file format, expand **Additional settings**, turn on **Speech file format settings**, choose the file format that you want (such as MP3, OGG, PCM, Mu-law, or A-law), and then choose **Download**.

AWS CLI

In this exercise, you call the `SynthesizeSpeech` operation by passing input text. You can save the resulting audio as a file and verify its content.

1. Run the `synthesize-speech` AWS CLI command to synthesize sample text to an audio file (`hello.mp3`).

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly synthesize-speech \  
  --output-format mp3 \  
  --voice-id Joanna \  
  --text 'Hello, my name is Joanna. I learned about the W3C on 10/3 of last  
year.' \  
  hello.mp3
```

In the call to `synthesize-speech`, you provide sample text to be synthesized by a voice of your choice. You must provide a voice ID (explained in the following step) and an output format. The command saves the resulting audio to the `hello.mp3` file. In addition to the MP3 file, the operation sends the following output to the console.

```
{  
    "ContentType": "audio/mpeg",  
    "RequestCharacters": "71"  
}
```

2. Play the resulting `hello.mp3` file to verify the synthesized speech.

Python

To test the Python example code, you need the AWS SDK for Python (Boto). For instruction, see [AWS SDK for Python \(Boto3\)](#).

The Python code in this example performs the following actions:

- Invokes the AWS SDK for Python (Boto) to send a `SynthesizeSpeech` request to Amazon Polly (by providing some text as input).
- Accesses the resulting audio stream in the response and saves the audio to a file (`speech.mp3`) on your local disk.
- Plays the audio file with the default audio player for your local system.

Save the code to a file (example.py) and run it.

```
"""Getting Started Example for Python 2.7+/3.3+"""
from boto3 import Session
from botocore.exceptions import BotoCoreError, ClientError
from contextlib import closing
import os
import sys
import subprocess
from tempfile import gettempdir

# Create a client using the credentials and region defined in the [adminuser]
# section of the AWS credentials file (~/.aws/credentials).
session = Session(profile_name="adminuser")
polly = session.client("polly")

try:
    # Request speech synthesis
    response = polly.synthesize_speech(Text="Hello world!", OutputFormat="mp3",
                                       VoiceId="Joanna")
except (BotoCoreError, ClientError) as error:
    # The service returned an error, exit gracefully
    print(error)
    sys.exit(-1)

# Access the audio stream from the response
if "AudioStream" in response:
    # Note: Closing the stream is important because the service throttles on the
    # number of parallel connections. Here we are using contextlib.closing to
    # ensure the close method of the stream object will be called automatically
    # at the end of the with statement's scope.
    with closing(response["AudioStream"]) as stream:
        output = os.path.join(gettempdir(), "speech.mp3")

        try:
            # Open a file for writing the output as a binary stream
            with open(output, "wb") as file:
                file.write(stream.read())
        except IOError as error:
            # Could not write to file, exit gracefully
            print(error)
            sys.exit(-1)
else:
```

```
# The response didn't contain audio data, exit gracefully
print("Could not stream audio")
sys.exit(-1)

# Play the audio using the platform's default player
if sys.platform == "win32":
    os.startfile(output)
else:
    # The following works on macOS and Linux. (Darwin = mac, xdg-open = linux).
    opener = "open" if sys.platform == "darwin" else "xdg-open"
    subprocess.call([opener, output])
```

For more in-depth examples, see the following topics:

- [Using SSML on the console](#)
- [Applying lexicons \(Synthesizing Speech\)](#)
- [Sample code and applications for Amazon Polly](#)

Synthesizing speech with bidirectional streaming

Amazon Polly provides a `StartSpeechSynthesisStream` operation that establishes an HTTP/2 connection with bidirectional communication between your application and the service. Text flows from your application to Amazon Polly while synthesized audio flows back. You send text as it becomes available, and Amazon Polly returns audio as it synthesizes, without either side waiting for the other to finish.

This is useful when text is produced progressively rather than all at once. For example, a customer service chatbot powered by a foundation model on Amazon Bedrock generates its response token by token. With bidirectional streaming, your application can forward each text chunk to Amazon Polly as the model produces it and begin playing audio back to the caller while the model is still generating the rest of the response.

This operation requires the generative engine and an AWS SDK that supports HTTP/2 event streams. The audio arrives as a sequence of chunks that your application accumulates into a complete audio output. Speech marks are not supported by this operation.

Note

The AWS CLI (v1 and v2), AWS Tools for PowerShell (v4 and v5), Python, and .NET v3 are not supported. You can use the bidirectional streaming API with the following SDKs: AWS SDK for Java 2.x, JavaScript v3, .NET v4, C++, Go v2, Kotlin, PHP v3, Ruby v3, Rust, and Swift.

Topics

- [SynthesizeSpeech and StartSpeechSynthesisStream compared](#)
- [Sending text and receiving audio](#)
- [Code examples](#)

SynthesizeSpeech and StartSpeechSynthesisStream compared

[SynthesizeSpeech](#) is a request-response operation. You provide the complete text in a single request and receive the full synthesized audio in a single response. It supports all engines

(standard, neural, long-form, generative), all output formats including speech marks, and has a text limit of 6,000 total characters (of which no more than 3,000 can be billed characters) per request. The response streams audio back as soon as the first bytes are available. Use this operation when you have all the text available upfront.

[StartSpeechSynthesisStream](#) is a bidirectional streaming operation. It opens an HTTP/2 connection over which you send text incrementally and receive audio as it is synthesized. There is no per-request text limit since text is streamed continuously. It requires the generative engine and does not support speech marks. Use this operation when text arrives incrementally and you want audio output to begin before all input is available. Common scenarios include:

- **Conversational AI and voice assistants.** A large language model generates response text in small chunks (tokens). Forward each text chunk to Amazon Polly as it arrives so the user hears speech while the model is still generating.
- **Real-time translation.** A translation system produces translated text segment by segment. Stream each segment for synthesis without waiting for the full translation to complete.
- **Long-form content exceeding SynthesizeSpeech limits.** Text longer than 6,000 characters can be streamed continuously without splitting into multiple requests or managing chunk boundaries.

Comparison of SynthesizeSpeech and StartSpeechSynthesisStream

Aspect	SynthesizeSpeech	StartSpeechSynthesisStream
Protocol	Request-response	Bidirectional event stream (HTTP/2)
Text delivery	Full text in request body	Streaming input text via TextEvent messages
Audio delivery	Streaming audio response via HTTP response body	Streaming audio response via AudioEvent messages
Engine support	standard, neural, long-form, generative	generative only
SSML support	Yes (all engines; Supported SSML tags)	Yes (Supported SSML tags)

Aspect	SynthesizeSpeech	StartSpeechSynthesisStream
Lexicons	Yes	Yes
Speech marks	Yes	No
Text limit	6,000 total characters (3,000 billed) per request	6,000 total characters (3,000 billed) per TextEvent
AWS CLI support	Yes	No (bidirectional streaming requires an SDK)

Sending text and receiving audio

A bidirectional streaming session involves opening a connection, sending text and receiving audio concurrently, then closing the stream when input is complete. The following sections describe each phase in detail.

Open the stream

Your application calls the [StartSpeechSynthesisStream](#) operation through the SDK, specifying synthesis parameters (Engine, VoiceId, OutputFormat, and optionally LanguageCode, LexiconNames, SampleRate). The SDK establishes an HTTP/2 connection and the bidirectional stream is ready to accept input events.

Send text

The client sends one or more [TextEvent](#) messages on the input stream. Each event can be sent as soon as text is available, without waiting for the full input to be ready. Text events do not need to align with sentence or punctuation boundaries. Amazon Polly reassembles the text internally and produces natural-sounding speech regardless of how the input is split across events.

Note

When using [Generating speech from SSML documents](#), each SSML document must be self-contained within a single TextEvent. You cannot split SSML tags across multiple events. However, you can mix plain text events and SSML events within the same stream.

The stream is subject to the following time limits:

- **Maximum stream duration:** 10 minutes. Amazon Polly closes the stream after 10 minutes regardless of activity. If your content requires more time, open a new stream for the remaining text.
- **Idle timeout between consecutive events:** 5 seconds. If no input event is sent for 5 seconds, Amazon Polly closes the stream. If your text source has pauses longer than 5 seconds, send a keep-alive `TextEvent` with an empty or whitespace string to prevent the timeout.

Forcing synthesis of buffered text with flushing

By default, Amazon Polly decides when to synthesize buffered text based on natural language boundaries. This produces the best audio quality but means audio may not be returned immediately after you send a `TextEvent`.

Flushing gives you control over when synthesis happens. When you flush, Amazon Polly immediately synthesizes all text it has buffered so far, regardless of whether the text ends at a natural boundary. This is useful when your text source pauses between logical sections and you want to deliver audio for what has been sent so far.

To flush, set the [FlushStreamConfiguration](#).`Force` parameter to `true` on a `TextEvent`. You can also send an empty `TextEvent` with the flush flag set to trigger synthesis without adding new content.

Flushing is a tradeoff. Setting `Force` to `true` mid-sentence may affect pronunciation and intonation because the synthesizer lacks context about what follows. For best results, allow Amazon Polly to buffer to natural boundaries whenever possible and only force synthesis when latency requirements demand it.

Receive audio

As Amazon Polly synthesizes text, it returns [AudioEvent](#) messages on the output stream. Each event contains a chunk of audio data. Your application must accumulate these chunks (for example, by writing them sequentially to a file or audio buffer) to produce the complete audio output. Audio events can arrive while you are still sending text events.

Close the stream

When all input text has been sent, the client sends a [CloseStreamEvent](#). Amazon Polly finishes processing any remaining buffered text, sends final audio events, and returns a [StreamClosedEvent](#) that contains the total number of characters synthesized. Always send a `CloseStreamEvent` rather than relying on flushing to end the stream. Closing ensures that all buffered text is synthesized and returned.

For full details on request parameters, event types, and errors, see the [StartSpeechSynthesisStream API reference](#).

Code examples

The following example demonstrates a complete bidirectional streaming session. The program creates an async Amazon Polly client, opens a stream configured with the generative engine and MP3 output, sends text as a series of `TextEvent` messages, accumulates the returned `AudioEvent` chunks into an output file, and closes the stream with a `CloseStreamEvent`. Because input and output happen concurrently, audio data begins arriving before all text has been sent.

Java

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyAsyncClient;
import software.amazon.awssdk.services.polly.model.*;

import java.io.FileOutputStream;
import java.io.OutputStream;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.Semaphore;
import org.reactivestreams.Publisher;
import org.reactivestreams.Subscription;

public class BidirectionalStreamExample {

    public static void main(String[] args) throws Exception {
        try (PollyAsyncClient pollyClient = PollyAsyncClient.builder()
            .region(Region.US_EAST_1)
            .build()) {
```



```

        StartSpeechSynthesisStreamRequest request =
StartSpeechSynthesisStreamRequest.builder()
    .engine(Engine.GENERATIVE)
    .voiceId(VoiceId.TIFFANY)
    .outputFormat(OutputFormat.MP3)
    .sampleRate("24000")
    .build();

    try (OutputStream audioOutput = new FileOutputStream("output.mp3")) {

        StartSpeechSynthesisStreamResponseHandler responseHandler =
            StartSpeechSynthesisStreamResponseHandler.builder()

                .subscriber(StartSpeechSynthesisStreamResponseHandler.Visitor.builder()
                    .onAudioEvent(audioEvent -> {
                        try {
                            byte[] audio =
audioEvent.audioChunk().asByteArray();
                            System.out.println("Received AudioEvent: " +
audio.length + " bytes");

                            audioOutput.write(audio);
                        } catch (Exception e) {
                            throw new RuntimeException(e);
                        }
                    })
                    .onStreamClosedEvent(closedEvent -> {
                        System.out.println("Stream closed. Characters
synthesized: "
                                + closedEvent.requestCharacters());
                    })
                    .onDefault(event -> {})
                    .build())
                .onError(error -> {
                    System.err.println("Stream error: " +
error.getMessage());
                })
                .build();

        String[] textChunks = {
            "The weather forecast for today shows clear skies ",
            "with temperatures reaching twenty five degrees. ",
            "Tomorrow we expect some cloud cover in the morning ",
            "but it should clear up by the afternoon. ",
            "The rest of the week looks mostly sunny ",

```

```

        "with a slight chance of rain on Friday. ",
        "Overall a great week to spend time outdoors."
    };

    Publisher<StartSpeechSynthesisStreamActionStream> inputPublisher =
subscriber -> {
    subscriber.onSubscribe(new Subscription() {
        private final Semaphore permits = new Semaphore(0);
        private volatile boolean cancelled = false;

        @Override
        public void request(long n) {
            permits.release((int) Math.min(n, Integer.MAX_VALUE));
        }

        @Override
        public void cancel() {
            cancelled = true;
            permits.release();
        }

        {
            new Thread(() -> {
                for (String chunk : textChunks) {
                    try { permits.acquire(); } catch (InterruptedException
e) { return; }

                    if (cancelled) return;
                    System.out.println("Sending TextEvent: " +
chunk.trim());

                    subscriber.onNext(StartSpeechSynthesisStreamActionStream.textEventBuilder()
                        .text(chunk).textType(TextType.TEXT).build());
                    // Simulate delay between chunks (e.g. waiting for LLM
tokens)
                    try { Thread.sleep(300); } catch (InterruptedException
e) { return; }

                }
                if (!cancelled) {
                    subscriber.onNext(StartSpeechSynthesisStreamActionStream
                        .closeStreamEventBuilder().build());
                    subscriber.onComplete();
                }
            }).start();
        }
    }
}

```

```

        });
    };

    CompletableFuture<Void> future = pollyClient.startSpeechSynthesisStream(
        request, inputPublisher, responseHandler);

    future.join();
} // audioOutput closed
} // pollyClient closed
}
}

```

JavaScript

```

import { PollyClient, StartSpeechSynthesisStreamCommand } from "@aws-sdk/client-polly";
import { createWriteStream } from "fs";

const client = new PollyClient({ region: "us-east-1" });

const textChunks = [
    "The weather forecast for today shows clear skies ",
    "with temperatures reaching twenty five degrees. ",
    "Tomorrow we expect some cloud cover in the morning ",
    "but it should clear up by the afternoon. ",
    "The rest of the week looks mostly sunny ",
    "with a slight chance of rain on Friday. ",
    "Overall a great week to spend time outdoors.",
];

const sleep = (ms) => new Promise((resolve) => setTimeout(resolve, ms));

async function* createInputEvents() {
    for (const chunk of textChunks) {
        console.log(`Sending TextEvent: ${chunk.trim()}`);
        yield {
            TextEvent: {
                Text: chunk,
                TextType: "text",
            },
        };
    }
    // Simulate delay between chunks (e.g. waiting for LLM tokens)
    await sleep(300);
}

```

```

    }

    yield { CloseStreamEvent: {} };
  }

  async function synthesizeStream() {
    const command = new StartSpeechSynthesisStreamCommand({
      Engine: "generative",
      VoiceId: "Tiffany",
      OutputFormat: "mp3",
      SampleRate: "24000",
      ActionStream: createInputEvents(),
    });

    const response = await client.send(command);
    const outputStream = createWriteStream("output.mp3");

    for await (const event of response.EventStream) {
      if (event.AudioEvent) {
        console.log(`Received AudioEvent: ${event.AudioEvent.AudioChunk.length}
bytes`);
        outputStream.write(event.AudioEvent.AudioChunk);
      } else if (event.StreamClosedEvent) {
        console.log(
          `Stream closed. Characters synthesized:
${event.StreamClosedEvent.RequestCharacters}`
        );
      }
    }

    outputStream.end();
  }

  synthesizeStream().catch(console.error);

```

Go

```

package main

import (
    "context"
    "fmt"
    "os"

```

```

"strings"
"time"

"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/config"
"github.com/aws/aws-sdk-go-v2/service/polly"
"github.com/aws/aws-sdk-go-v2/service/polly/types"
)

func main() {
    ctx := context.Background()

    cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion("us-east-1"))
    if err != nil {
        fmt.Fprintf(os.Stderr, "Failed to load config: %v\n", err)
        os.Exit(1)
    }

    client := polly.NewFromConfig(cfg)

    output, err := client.StartSpeechSynthesisStream(ctx,
        &polly.StartSpeechSynthesisStreamInput{
            Engine:      types.EngineGenerative,
            VoiceId:    types.VoiceIdTiffany,
            OutputFormat: types.OutputFormatMp3,
            SampleRate: aws.String("24000"),
        })
    if err != nil {
        fmt.Fprintf(os.Stderr, "Failed to start stream: %v\n", err)
        os.Exit(1)
    }

    stream := output.GetStream()
    defer stream.Close()

    audioFile, err := os.Create("output.mp3")
    if err != nil {
        fmt.Fprintf(os.Stderr, "Failed to create output file: %v\n", err)
        os.Exit(1)
    }
    defer audioFile.Close()

    textChunks := []string{
        "The weather forecast for today shows clear skies ",
    }

```

```

    "with temperatures reaching twenty five degrees. ",
    "Tomorrow we expect some cloud cover in the morning ",
    "but it should clear up by the afternoon. ",
    "The rest of the week looks mostly sunny ",
    "with a slight chance of rain on Friday. ",
    "Overall a great week to spend time outdoors.",
}

// Send text events in a goroutine
go func() {
    for _, chunk := range textChunks {
        fmt.Printf("Sending TextEvent: %s\n", strings.TrimSpace(chunk))
        err := stream.Send(ctx,
&types.StartSpeechSynthesisStreamActionStreamMemberTextEvent{
            Value: types.TextEvent{
                Text:      aws.String(chunk),
                TextType: types.TextTypeText,
            },
        })
        if err != nil {
            fmt.Fprintf(os.Stderr, "Failed to send text event: %v\n", err)
            return
        }
        // Simulate delay between chunks (e.g. waiting for LLM tokens)
        time.Sleep(300 * time.Millisecond)
    }

    // Signal end of input
    stream.Send(ctx,
&types.StartSpeechSynthesisStreamActionStreamMemberCloseStreamEvent{
        Value: types.CloseStreamEvent{},
    })
}()

// Receive audio events
for event := range stream.Events() {
    switch v := event.(type) {
    case *types.StartSpeechSynthesisStreamEventStreamMemberAudioEvent:
        fmt.Printf("Received AudioEvent: %d bytes\n", len(v.Value.AudioChunk))
        audioFile.Write(v.Value.AudioChunk)
    case *types.StartSpeechSynthesisStreamEventStreamMemberStreamClosedEvent:
        fmt.Printf("Stream closed. Characters synthesized: %d\n",
v.Value.RequestCharacters)
    }
}

```

```

}

if err := stream.Err(); err != nil {
    fmt.Fprintf(os.Stderr, "Stream error: %v\n", err)
    os.Exit(1)
}
}

```

Rust

```

use aws_sdk_polly::types::{
    CloseStreamEvent, Engine, OutputFormat,
    StartSpeechSynthesisStreamActionStream,
    StartSpeechSynthesisStreamEventStream, TextEvent, TextType, VoiceId,
};
use aws_sdk_polly::Client;
use std::fs::File;
use std::io::Write;
use tokio::time::{sleep, Duration};

#[tokio::main]
async fn main() {
    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .region("us-east-1")
        .load()
        .await;

    let client = Client::new(&config);

    let text_chunks = vec![
        "The weather forecast for today shows clear skies ",
        "with temperatures reaching twenty five degrees. ",
        "Tomorrow we expect some cloud cover in the morning ",
        "but it should clear up by the afternoon. ",
        "The rest of the week looks mostly sunny ",
        "with a slight chance of rain on Friday. ",
        "Overall a great week to spend time outdoors.",
    ];

    let input_stream = async_stream::stream! {
        for chunk in &text_chunks {
            println!("Sending TextEvent: {}", chunk.trim());
            yield Ok(StartSpeechSynthesisStreamActionStream::TextEvent(

```

```

TextEvent::builder().text(*chunk).text_type(TextType::Text).build().unwrap(),
    ));
    // Simulate delay between chunks (e.g. waiting for LLM tokens)
    sleep(Duration::from_millis(300)).await;
}
yield Ok(StartSpeechSynthesisStreamActionStream::CloseStreamEvent(
    CloseStreamEvent::builder().build(),
));
};

let mut output = client
    .start_speech_synthesis_stream()
    .engine(Engine::Generative)
    .voice_id(VoiceId::Tiffany)
    .output_format(OutputFormat::Mp3)
    .sample_rate("24000")
    .action_stream(input_stream.into())
    .send()
    .await
    .expect("Failed to start stream");

let mut audio_file = File::create("output.mp3").expect("Failed to create output
file");

while let Ok(Some(event)) = output.event_stream.recv().await {
    match event {
        StartSpeechSynthesisStreamEventStream::AudioEvent(audio_event) => {
            if let Some(chunk) = audio_event.audio_chunk() {
                let bytes = chunk.as_ref();
                println!("Received AudioEvent: {} bytes", bytes.len());
                audio_file.write_all(bytes).unwrap();
            }
        }
        StartSpeechSynthesisStreamEventStream::StreamClosedEvent(closed_event)
=> {
            println!(
                "Stream closed. Characters synthesized: {}",
                closed_event.request_characters()
            );
        }
        _ => {}
    }
}
}

```



```
}
```

Voices in Amazon Polly

Amazon Polly provides dozens of lifelike voices and support for a variety of languages. Each voice is created using native language speakers, so there are variations from voice to voice, even within the same language. You can also use the AWS Management Console to test each voice with text of your choice. For most languages, there will be at least one male and one female voice, and often more than one of each. A few languages only have a single voice.

The inventory of voices and the number of languages included is continually being updated to include additional choices. To suggest a new language or voice, provide feedback on this page. Unfortunately, we are not able to comment on plans for specific new languages before they are released.

Note

To hear example Amazon Polly voices in your browser, see the [Amazon Polly product overview](#).

Topics

- [Available voices](#)
- [Bilingual voices](#)
- [Applying the newscaster voice](#)
- [Listening to voices](#)
- [Timing a voice speed](#)
- [Changing a voice speed](#)

Available voices

Amazon Polly provides a variety of lifelike voices in multiple languages for synthesizing speech from text. The following table shows all the voices that Amazon Polly offers.

	Language and language variants	Language code	Name/ ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
1	Arabic	arb	Zeina	Female	No	No	No	Yes
2	Arabic (Gulf)	ar-AE	Hala*	Female	No	No	Yes	No
			Zayd*	Male	No	No	Yes	No
3	Dutch (Belgian)	nl-BE	Lisa	Female	Yes	No	Yes	No
4	Catalan	ca-ES	Arlet	Female	No	No	Yes	No
5	Czech	cs-CZ	Jitka	Female	No	No	Yes	No
6	Chinese (Cantonese)	yue-CN	Hiujin	Female	No	No	Yes	No
7	Chinese (Mandarin)	cmn-CN	Zhiyu	Female	No	No	Yes	Yes
8	Danish	da-DK	Naja	Female	No	No	No	Yes
			Mads	Male	No	No	No	Yes
			Sofie	Female	No	No	Yes	No
9	Dutch	nl-NL	Laura	Female	Yes	No	Yes	No
			Lotte	Female	No	No	No	Yes
			Ruben	Male	No	No	No	Yes
10	English (Australian)	en-AU	Nicole	Female	No	No	No	Yes
			Olivia	Female	Yes	No	Yes	No

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
			Russell	Male	No	No	No	Yes
11	English (British)	en-GB	Amy**	Female	Yes	No	Yes	Yes
			Emma	Female	No	No	Yes	Yes
			Brian	Male	Yes	No	Yes	Yes
			Arthur	Male	No	No	Yes	No
12	English (Indian)	en-IN	Aditi*	Female	No	No	No	Yes
			Raveena	Female	No	No	No	Yes
			Kajal*	Female	Yes	No	Yes	No
13	English (Ireland)	en-IE	Niamh	Female	Yes	No	Yes	No
14	English (New Zealand)	en-NZ	Aria	Female	Yes	No	Yes	No
15	English (Singaporean)	en-SG	Jasmine	Female	Yes	No	Yes	No
16	English (South African)	en-ZA	Ayanda	Female	Yes	No	Yes	No

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
17	English (US)	en-US	Danielle	Female	Yes	Yes	Yes	No
			Gregory	Male	No	Yes	Yes	No
			Ivy	Female	No	No	Yes	Yes
			Joanna**	(child)	Yes	No	Yes	Yes
			Kendra	Female	No	No	Yes	Yes
			Kimberly	Female	No	No	Yes	Yes
			Salli	Female	Yes	No	Yes	Yes
			Joey	Female	No	No	Yes	Yes
			Justin	Male	No	No	Yes	No
			Kevin	Male	No	No	Yes	Yes
			Matthew*	(child)	Yes	No	Yes	No
			Ruth	(child)	Yes	Yes	Yes	No
			Stephen	Male	Yes	No	Yes	No
			Tiffany	Female	Yes	No	No	No
			Patrick	Male	No	Yes	No	No
				Female				
				Male				
18	English (Welsh)	en-GB-WLS	Geraint	Male	No	No	No	Yes

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
19	Finnish	fi-FI	Suvi	Female	No	No	Yes	No
20	French	fr-FR	Ambre	Female	Yes	No	No	No
			Céline/ Celine	Female	Yes	No	No	Yes
				Male	Yes	No	No	No
			Florian	Female	Yes	No	Yes	Yes
			Léa	Male	No	No	No	Yes
			Mathieu	Male	Yes	No	Yes	No
			Rémi					
21	French (Belgian)	fr-BE	Isabelle	Female	Yes	No	Yes	No
22	French (Canadian)	fr-CA	Chantal	Female	No	No	No	Yes
			Gabrielle	Female	Yes	No	Yes	No
			Liam	Male	Yes	No	Yes	No
23	German	de-DE	Marlene	Female	No	No	No	Yes
			Vicki	Female	Yes	No	Yes	Yes
			Hans	Male	No	No	No	Yes
			Daniel	Male	Yes	No	Yes	No
			Lennart	Male	Yes	No	No	No

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
24	German (Austrian)	de-AT	Hannah	Female	Yes	No	Yes	No
25	German (Swiss)	de-CH	Sabrina	Female	Yes	No	Yes	No
26	Hindi	hi-IN	Aditi*	Female	No	No	No	Yes
			Kajal*	Female	No	No	Yes	No
27	Icelandic	is-IS	Dóra/ Dora	Female	No	No	No	Yes
			Karl	Male	No	No	No	Yes
28	Italian	it-IT	Beatrice	Female	Yes	No	No	No
			Carla	Female	No	No	No	Yes
			Bianca	Female	Yes	No	Yes	Yes
			Lorenzo	Male	Yes	No	No	No
			Giorgio	Male	No	No	No	Yes
			Adriano	Male	No	No	Yes	No
29	Japanese	ja-JP	Mizuki	Female	No	No	No	Yes
			Takumi	Male	No	No	Yes	Yes
			Kazuha	Female	No	No	Yes	No
			Tomoko	Female	No	No	Yes	No

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
30	Korean	ko-KR	Seoyeon	Female	Yes	No	Yes	Yes
			Jihye	Female	No	No	Yes	No
31	Norwegian	nb-NO	Liv	Female	No	No	No	Yes
			Ida	Female	No	No	Yes	No
32	Polish	pl-PL	Ewa	Female	Yes	No	No	Yes
			Maja	Female	No	No	No	Yes
			Jacek	Male	No	No	No	Yes
			Jan	Male	No	No	No	Yes
			Ola	Female	Yes	No	Yes	No
33	Portuguese (Brazilian)	pt-BR	Camila	Female	Yes	No	Yes	Yes
			Vitória/Vitoria	Female	No	No	Yes	Yes
				Male	No	No	No	Yes
			Ricardo	Male	No	No	Yes	No
			Thiago					
34	Portuguese (European)	pt-PT	Inês/Ines	Female	No	No	Yes	Yes
			Cristiano	Male	No	No	No	Yes
35	Romanian	ro-RO	Carmen	Female	No	No	No	Yes

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
36	Russian	ru-RU	Tatyana	Female	No	No	No	Yes
			Maxim	Male	No	No	No	Yes
37	Spanish (Spain)	es-ES	Conchita	Female	No	No	No	Yes
			Lucia	Female	Yes	No	Yes	Yes
			Alba	Female	No	Yes	No	No
			Enrique	Male	No	No	No	Yes
			Sergio	Male	Yes	No	Yes	No
			Raúl	Male	No	Yes	No	No
38	Spanish (Mexican)	es-MX	Mia	Female	Yes	No	Yes	Yes
			Andrés	Male	Yes	No	Yes	No
39	Spanish (US)	es-US	Lupe**	Female	Yes	No	Yes	Yes
			Penélope	Female	No	No	No	Yes
			Penelope	Male	No	No	No	Yes
			Miguel	Male	Yes	No	Yes	No
			Pedro					
40	Swedish	sv-SE	Astrid	Female	No	No	No	Yes
			Elin	Female	No	No	Yes	No
41	Turkish	tr-TR	Filiz	Female	No	No	No	Yes
			Burcu	Female	No	No	Yes	No

	Language and language variants	Language code	Name/ID	Gender	Generative voice	Long Form voice	Neural voice	Standard voice
42	Welsh	cy-GB	Gwyneth	Female	No	No	No	Yes

* This voice is bilingual. For more information, see [Bilingual voices](#).

** These voices can be used with Newscaster speaking styles when used with the Neural format. For more information, see [Applying the newscaster voice](#).

Each Amazon Polly voice engine has unique features. Learn more about features and Region availability for the voice engines offered by Amazon Polly:

- [Generative voices](#)
- [Long-form voices](#)
- [Neural voices](#)
- [Standard voices](#)

Brand voices

In addition to the available voices listed in the previous table, you can use Amazon Polly to build a custom voice for your brand persona. With a brand voice, you can offer unique and exclusive voices to your customers. To learn more about Amazon Polly brand voices, see [Brand Voice](#).

Bilingual voices

Amazon Polly has two ways of producing bilingual voices:

- [Accented bilingual voices](#)
- [Fully bilingual voices](#)

Accented bilingual voices

Accented bilingual voices can be created using any Amazon Polly voice, but only when using SSML tags.

Normally, all words in the input text are spoken in the default language of the voice specified you're using.

For example, if you're using the voice of Joanna (who speaks US English), Amazon Polly speaks the following in the Joanna voice without a French accent:

```
< speak >
  Why didn't she just say, 'Je ne parle pas français?'
< / speak >
```

In this case, the words *Je ne parle pas français* are spoken as they would be if they were English.

However, if you use the Joanna voice with the `<lang>` tag, Amazon Polly speaks the sentence in the Joanna voice in American-accented French:

```
< speak >
  Why didn't she just say, < lang xml:lang="fr-FR" > 'Je ne parle pas français?' < /
lang > .
< / speak >
```

Because Joanna is not a native French voice, pronunciation is based on her native language, US English. For instance, although perfect French pronunciation features an uvular trill /R/ in the word *français*, Joanna's US English voice pronounces this phoneme as the corresponding sound /r/.

If you use the voice of Giorgio, who speaks Italian, with the following text, Amazon Polly speaks the sentence in Giorgio's voice with an Italian pronunciation:

```
< speak >
  Mi piace Bruce Springsteen.
< / speak >
```

Fully bilingual voices

A fully bilingual voice like Aditi or Kajal (Indian English and Hindi) can speak two languages fluently. This gives you the ability to use words and phrases from both languages in a single text using the same voice.

Currently, Aditi, Kajal, Hala, and Zayd are the only fully bilingual voices available.

Using a Bilingual Voice (example: Aditi)

Aditi speaks both Indian English (en-IN) and Hindi (hi-IN) fluently. You can synthesize speech in both English and Hindi, and the voice can switch between the two languages even within the same sentence.

Hindi can be used in two different forms:

- Devanagari: "उसेन कहाँ, खेल तोह अब शुरू होगा"
- Romanagari (using the Latin alphabet): "Usne kahan, khel toh ab shuru hoga"

Additionally, it's possible to mix English and Hindi of either or both forms within a single sentence:

- Devanagari + English: "This is the song कभी कभी अदिति"
- Romanagari + English: "This is the song from the movie Jaane Tu Ya Jaane Na."
- Devanagari + Romanagari + English: "This is the song कभी कभी अदिति from the movie Jaane Tu Ya Jaane Na."

Because Aditi is a bilingual voice, text in all of these cases will be read correctly, as Amazon Polly can differentiate between the languages and scripts.

Amazon Polly also supports numbers, dates, times, and currency expansion in both English (Arabic numerals) and Hindi (Devanagari numerals). By default, Arabic numerals are read in Indian English. To make Amazon Polly read them in Hindi, you must use the `hi-IN` language code parameter.

Applying the newscaster voice

People use different speaking styles, depending on context. Casual conversation, for example, sounds very different from a TV or radio newscast. Because of the way standard voices are made, they can't produce different speaking styles. However, neural voices can. They can be trained for a specific speaking style, with the variations and emphasis on certain parts of speech inherent in that style.

In addition to the default neural voices, Amazon Polly provides a newscaster speaking style that uses the neural system to generate speech in the style of a TV or radio newscaster. The Newscaster

style is available with the Matthew and Joanna voices in US English (en-US), the Lupe voice in US Spanish (es-US), and the Amy voice in British English (en-GB).

To use the Newscaster style, first choose the neural engine and then use the syntax described in the following steps in your input text.

Note

- To use any neural speaking style, you must use one of the AWS Regions that support neural voices. This option is not available in all Regions. For more information, see [Feature and region compatibility](#).

Console

To apply the Newscaster style

1. Open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Make sure that you are using an AWS Region where neural voices are supported.
3. On the Text-to-Speech page, for **Engine**, choose **Neural**.
4. Choose the language and voice you want to use. Only Matthew and Joanna for US English (en-US), Lupe for US Spanish (es-US), and Amy for British English (en-GB) are available in the newscaster voice.
5. Turn on **SSML**.
6. Add input text to your text-to-speech request using the Newscaster style SSML syntax.

```
<amazon:domain name="news">text</amazon:domain>
```

For example, you might use the newscaster tag as follows:

```
<speack>
<amazon:domain name="news">
From the Tuesday, April 16th, 1912 edition of The Guardian newspaper:

The maiden voyage of the White Star liner Titanic, the largest ship ever
launched
ended in disaster.
```

```
The Titanic started her trip from Southampton for New York on Wednesday. Late
on
Sunday night she struck an iceberg off the Grand Banks of Newfoundland. By
wireless telegraphy she sent out signals of distress, and several liners were
near enough to catch and respond to the call.
</amazon:domain>
</speak>
```

7. Choose **Listen**.

AWS CLI

To apply the Newscaster style

1. In your API request, include the engine parameter with the `neural` value:

```
--engine neural
```

2. Add input text to your API request using the Newscaster style SSML syntax.

```
<amazon:domain name="news">text</amazon:domain>
```

For example, you might use the newscaster tag as follows:

```
<speak>
<amazon:domain name="news">
From the Tuesday, April 16th, 1912 edition of The Guardian newspaper:

The maiden voyage of the White Star liner Titanic, the largest ship ever
launched
ended in disaster.

The Titanic started her trip from Southampton for New York on Wednesday. Late
on
Sunday night she struck an iceberg off the Grand Banks of Newfoundland. By
wireless telegraphy she sent out signals of distress, and several liners were
near enough to catch and respond to the call.
</amazon:domain>
</speak>
```

For more information about SSML, see [Supported SSML tags](#).

Listening to voices

Once you have [set up](#) Amazon Polly, you can test voices using custom text on the console.

To listen to Amazon Polly voices on the console

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Text-to-Speech** tab.
3. For **Engine**, choose **Generative, Long Form, Neural**, or **Standard**.
4. Select a language and a Region. Then choose a voice.
5. Enter text for the voice to speak or use the default phrase, and then choose **Listen**.

Timing a voice speed

Because of the natural variation between voices, each available voice speaks at slightly different speeds. For instance, with US English voices, Ivy and Joanna are slightly faster than Matthew, and considerably faster than Joey. Since there is so much variation between voices, there is no standard speed (words per minute) available for Amazon Polly voices. However, you can find how long it takes for your voice to say the selected text using [Speech Marks](#).

To time the length of a spoken text passage

1. Open the AWS CLI.
2. Run the following code, filling in as needed.

```
aws polly synthesize-speech \  
  --language-code optional language code if needed \  
  --output-format json \  
  --voice-id [name of desired voice] \  
  --text '[desired text]' \  
  --speech-mark-types='["viseme"]' \  
  LengthOfText.txt
```

3. Open LengthOfText.txt.

If the text were "Mary had a little lamb," the last few lines returned by Amazon Polly would be:

```
{"time":882,"type":"viseme","value":"t"}  
{"time":964,"type":"viseme","value":"a"}  
{"time":1082,"type":"viseme","value":"p"}
```

The last viseme, essentially the sound for the final letters in "lamb" starts 1082 milliseconds after the beginning of the speech. While this is not exactly the length of the audio, it's close and can serve as the basis for comparison between voices.

Changing a voice speed

For certain applications, you may find that you'd prefer the voice you like be slowed down, or speeded up. If the speed of the voice is a concern, Amazon Polly provides the ability to modify this using SSML tags. For example, if your organization was making an application that reads books to immigrant audiences, you may want to vary the voice speed. Your audience may speak English, but their fluency is limited. Amazon Polly helps you slow down the rate of speech using the SSML `<prosody>` tag.

You can use a percentage:

```
<speak>  
  In some cases, it might help your audience to <prosody rate="85%">slow  
  the speaking rate slightly to aid in comprehension.</prosody>  
</speak>
```

Or a preset speed:

```
<speak>  
  In some cases, it might help your audience to <prosody rate="slow">slow  
  the speaking rate slightly to aid in comprehension.</prosody>  
</speak>
```

Two speed options are available to you when using SSML with Amazon Polly:

- **Preset speeds:** x-slow, slow, medium, fast, and x-fast. In these cases, the speed of each option is approximate, depending on your preferred voice. The medium option is the normal speed of the voice.

- **n% of speech rate:** any percentage of the speech rate, between 20% and 200% can be used. In these cases, you can choose exactly the speed you want. However, the actual speed of the voice is approximate, depending on the voice you've chosen. 100% is considered to be the normal speed of the voice.

 Note

Test your selected voice at various speeds. The speed of each option is approximate and depends on the voice you choose.

For more information on using the prosody tag, see [Controlling volume, speaking rate, and pitch](#).

Languages in Amazon Polly

The following languages are supported by Amazon Polly and can be used to synthesize speech. Each language has a unique language code. These language codes are [W3C language identification tags](#) (*ISO 639-3* for the language name and *ISO 3166* for the country code).

Select a language from the following table for details on the phonemes and visemes that Amazon Polly provides.

Language	Language code
Arabic	arb
Arabic (Gulf)	ar-AE
Catalan	ca-ES
Chinese (Cantonese)	yue-CN
Chinese (Mandarin)	cmn-CN
Czech	cs-CZ
Danish	da-DK
Dutch (Belgian)	nl-BE
Dutch	nl-NL

Language	Language code	
English (Australian)	en-AU	
English (British)	en-GB	
English (Indian)	en-IN	
English (New Zealand)	en-NZ	
English (Singaporean)	en-SG	
English (South African)	en-ZA	
English (US)	en-US	
English (Welsh)	en-GB-WLS	
Finnish	fi-FI	
French	fr-FR	
French (Belgian)	fr-BE	

Language	Language code	
French (Canada)	fr-CA	
Hindi	hi-IN	
German	de-DE	
German (Austria)	de-AT	
German (Swiss standard)	de-CH	
Icelandic	is-IS	
Italian	it-IT	
Japanese	ja-JP	
Korean	ko-KR	
Norwegian	nb-NO	
Polish	pl-PL	
Portuguese (Brazilian)	pt-BR	

Language	Language code	
Portuguese (European)	pt-PT	
Romanian	ro-RO	
Russian	ru-RU	
Spanish (Spain)	es-ES	
Spanish (Mexico)	es-MX	
Spanish (US)	es-US	
Swedish	sv-SE	
Turkish	tr-TR	
Welsh	cy-GB	

Arabic (arb)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Arabic voice of Zeina that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ʔ	ʔ	glottal stop	أنا	
ʕ	ʔ\	voiced pharyngeal fricative	عَمَر	k
b	b	voiced bilabial plosive	بَلَد	p
d	d	voiced alveolar plosive	داري	t
dʕ	d_ʔ\	emphatic voiced alveolar plosive	ضَوء	t
ɗʒ	dʒ	voiced postalveolar affricate	جَمِيل	S
ð	D	voiced dental fricative	ذَلِكَ	T
ðʕ	D_ʔ\	emphatic voiced dental fricative	طَالَام	T
f	f	voiceless labiodental fricative	فَصَل	f
g	g	voiced velar plosive	إنجلترا	k
ɣ	G	voiced velar fricative	عَرَب	k
h	h	voiceless glottal fricative	هذا	k

IPA	X-SAMPA	Description	Example	Viseme
j	j	palatal approximant	يَمَشِي	i
k	k	voiceless velar plosive	كَلَب	k
l	l	alveolar lateral approximant	لَاقِي	t
l̤	l_G	emphatic alveolar lateral approximant	عَبْدَالله	t
m	m	bilabial nasal	مَازَا	p
n	n	alveolar nasal	نَوْر	t
p	p	voiceless bilabial plosive	حَبَسَ	p
q	q	voiceless uvular plosive	قَرِيْب	k
r	r	alveolar trill	رَمَل	r
s	s	voiceless alveolar fricative	سُؤَال	s
s̤	s_?\\	emphatic voiceless alveolar fricative	صَاحِب	s
ʃ	S	voiceless postalveolar fricative	شُكْر	S
t	t	voiceless alveolar plosive	تَمَر	t

IPA	X-SAMPA	Description	Example	Viseme
tʃ	t_?\\	emphatic voiceless alveolar plosive	طالِب	t
θ	T	voiceless dental fricative	ثَلَاث	T
v	v	voiced labiodental fricative	فِي تَامِيْن	f
w	w	labio-velar approximant	وَلَد	u
x	x	voiceless velar fricative	خَوْف	k
ħ	X\\	voiceless pharyngeal fricative	خَوْل	k
z	z	voiced alveolar fricative	زُهُور	s
Vowels				
a	a	open front unrounded vowel	بَرْد	a
a:	a:	long open front unrounded vowel	دَار	a
ɑʃ	A_?\\	emphatic open back unrounded vowel	طَابَل	a
ɑʃ:	A_?\\:	emphatic long open back unrounded vowel	ظَالِم	a

IPA	X-SAMPA	Description	Example	Viseme
u	u	close back rounded vowel	شُرْب	u
u:	u:	long close back rounded vowel	سور	u
u ^ʁ	u_?\ 	emphatic close back rounded vowel	بُدْ	u
u ^ʁ :	u_?\ :	emphatic long close back rounded vowel	طول	u
i	i	close front unrounded vowel	بِنْت	i
i:	i:	long close front unrounded vowel	حَزِين	i
i ^ʁ	i_?\ 	emphatic close front unrounded vowel	ضَدْ	i
i ^ʁ :	i_?\ :	emphatic long close front unrounded vowel	ماضي	i
e	e	close-mid front unrounded vowel	ماركْت	e
e:	e:	long close-mid front unrounded vowel	موديل	e
ɔ	O	open-mid back rounded vowel	تكنولوجيا	O

IPA	X-SAMPA	Description	Example	Viseme
ɔ:	O:	long open-mid back rounded vowel	تلفزيون	O

Arabic (Gulf) (ar-AE)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Arabic voice of Hala that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
Consonants					
b	b	voiced bilabial plosive	ب	/ " b a . l a d /	b
d	d	voiced alveolar plosive	د	/ " r a d d /	d
d ^ʕ	d_?\\	pharyngea lised voiced alveolar plosive	ضوء	/ " d_?\\ a w ? /	D
f	f	voiceless labiodental fricative	فرن	/ " f l . r l n /	f
g	g	voiced velar plosive	قال	/ " g a : l /	k
j	j	voiced palatal approximant	يمني	/ " j l m . S i : /	i

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
k	k	voiceless velar plosive	كامل	/ " k a : . m i l /	k
l	l	voiced alveolar lateral approximant	لېل	/ " l e : l /	t
l̤	l_G	pharyngea lised voiced alveolar lateral approximant	عبدالله	/ ? \ a b . " d A_? \ l_G . l_G A_? \ /	t
m	m	bilabial nasal stop	مئة	/ " m l j . j a /	p
n	n	alveolar nasal stop	نور	/ " n u : r /	t
p	p	voiceless bilabial plosive	أوبرا	/ " ? O . p e . r a : /	p
q	q	voiceless uvular plosive	قصر	/ " q A_? \ s_? \ r /	k
r	r	alveolar trill	رمل	/ " r a . m i l l /	r
s	s	voiceless alveolar fricative	سمسم	/ " s l m . s l m /	s
s̤	s_? \	pharyngea lised voiceless alveolar fricative	صاحب	/ " s_? \ A_?: . X \ l b /	s

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
t	t	voiceless alveolar plosive	تمر	/ "t a . m a r /	t
tʕ	t_? \	pharyngea lised voiceless alveolar fricative	طالب	/ " t_? \ A_?: . l l b /	t
v	v	voiced labiodental fricative	فيتامين	/ v i: . t A . " m i: n /	f
w	w	voiced labiovelar approximant	وايد	/ " w a: . j l d /	u
x	x	voiceless velar fricative	خروف	/ x a . " r u: f /	k
z	z	voiceless velar fricative	زهور	/ " z h u: r /	s
ð	D	voiced interdental fricative	ذلك	/ " D a: . l l k /	D
ðʕ	D_? \	pharyngea lised voiced interdental fricative	ظلام	/ D_? \ A_? \ . " l a: m /	D
ħ	X \	voiceless pharyngeal fricative	الحين	/ ? a l . " X \ i: n /	k

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
ŋ	N	velar nasal stop	هونغ كونغ	/ h O N . " k O N g /	k
ɣ	G	voiced velar fricative	غريبة	/ G l . " r i : . b a /	k
ʃ	S	voiceless postalveolar fricative	شمس	/ " S a m s /	S
ʒ	Z	voiced postalveolar fricative	جالكيت	/ Z a . " k e : t /	S
ʔ	ʔ	glottal stop	مؤسسة	/ m u . " ʔ a s . s a . s a /	
ʕ	ʔ\	voiced pharyngeal fricative	عام	/ " ʔ\ a : m m /	k
dʒ	dZ	voiced postalveolar affricate	جامعة	/ " dZ a : m . ʔ\ a /	S
θ	T	voiced interdental fricative	ثلاثة	/ T a . " l a : . T a /	T
ħ	h	voiced glottal fricative	هلال	/ " h l a : l /	k
Vowels					

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
æ	a	mid-open front unrounded short vowel	سفر	/ " s a . f a r /	a
ɑ ^ʕ	A_?\ b /	pharyngea lised open back unrounded short vowel	صلب	/ " s_?\ A_?\ l b /	a
æ:	a:	mid-open front unrounded long vowel	باب	/ " b a: b /	a
ɑ ^ʕ :	A_?: dZ /	pharyngea lised open back unrounded long vowel	ناضج	/ " n A_?: . D_? \ i_?\ dZ /	a
a	A	open central unrounded short vowel	wifi	/ " w A j . f A j /	a
i	i	tense close front unrounded short vowel (MSA)	إسحاق	/ ? i s . " X\ A_? \ q /	i
ɪ	ɪ	lax close front unrounded short vowel	بنت	/ " b ɪ n t /	i

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
i ^ʕ	i_?\	pharyngea lised close front unrounded short vowel	طفل	/ " t_?\ i_?\ f l l /	i
i:	i:	close front unrounded long vowel	سبيل	/ s a . " b i: l /	i
i ^ʕ :	i_ ^ʕ :	pharyngea lised close front unrounded long vowel	رطيّب	/ r A_?\ . " t_?\ i_ ^ʕ : b /	i
u	u	tense close back rounded short vowel (MSA)	مخترع	/ " m u x . t a . r i ?\ /	u
ʊ	U	lax close back rounded short vowel	رسوم	/ r U . " s u: m /	u
u ^ʕ	u_?\	pharyngea lised close back rounded short vowel	عصفور	/ ?\ u_?\ s_?\ . " f u: r /	u
u:	u:	close back rounded long vowel	توت	/ " t u: t /	u

IPA	X-SAMPA	Description	Example	Pronunciation	Viseme
uː	u_?\":	pharyngea lised close back rounded long vowel	صَوْر	/ " s_?\" u_?\": r /	u
e	e	mid front unrounded short vowel	إِنْتَرَنْت	/ " s e n t /	e
e:	e:	mid front unrounded long vowel	إِش	/ " ? e: S /	e
ɔ	O	open-mid back rounded short vowel	دولار	/ d O . " l A r /	O
ɔ:	O:	open-mid back rounded long vowel	لون	/ " l O: n /	O

Catalan (ca-ES)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Catalan voice of Arlet that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
p	p	voiceless bilabial plosive	ploure	p

IPA	X-SAMPA	Description	Example	Viseme
t	t	voiceless alveolar plosive	Tarragona	t
k	k	voiceless velar plosive	com	k
b	b	voiced bilabial plosive	bata	p
d	d	voiced alveolar plosive	endoll	t
g	g	voiced velar plosive	gros	k
m	m	voiced bilabial nasal	manera	p
n	n	voiced alveolar nasal	donar	t
ɲ	J	voiced palatal nasal	any	J
ŋ	N	voiced velar nasal	pingüí	k
ɫ	5	voiced velarized alveolar lateral approximant (dark l)	albercoc	l
ʎ	L	voiced palatal lateral approximant	llop	J
r	r	voiced alveolar trill	parra	r
ɾ	4	voiced alveolar tap	para	t

IPA	X-SAMPA	Description	Example	Viseme
f	f	voiceless labiodental fricative	èmfasi	f
s	s	voiceless alveolar fricative	sac	s
z	z	voiced alveolar fricative	calzes	s
ʃ	S	voiceless postalveolar fricative	guix	S
ʒ	Z	voiced postalveolar fricative	col·legi	S
tʃ	tS	voiceless postalveolar affricate	cotxe	S
dʒ	dZ	voiced postalveolar affricate	platja	S
β	B	voiced bilabial approximant	obert	B
ð	D	voiced dental approximant	bedoll	T
j	j	voiced palatal approximant	noia	i
ɣ	G	voiced velar approximant	pega	k
v	v	voiced labiodental fricative	afgà	f
w	w	voiced labiovelar approximant	aigua	u

IPA	X-SAMPA	Description	Example	Viseme
x	x	voiceless velar fricative	Jiménez	k
j	j\	voiced palatal fricative	yeso	J
l	l	voiced alveolar lateral approximant	alondra	t
θ	T	voiceless dental fricative	González	T
Vowels				
a	a	open back vowel	casa	a
e	e	close-mid front unrounded vowel	llenya	e
ɛ	E	open-mid front unrounded vowel	xec	E
i	i	closed front unrounded vowel	visca	i
o	o	close-mid back rounded vowel	gos	o
ɔ	O	open-mid back rounded vowel	joc	O
u	u	closed back rounded vowel	un	u
ə	@	mid-central vowel	casa	@
Additional Symbols				

IPA	X-SAMPA	Description	Example	Viseme
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Chinese (Cantonese) (yue-CN)

The following table lists the Jyutping and International Phonetic Alphabet (IPA) phonemes for the Cantonese voice that is supported by Amazon Polly. Jyutping is a romanization system of Cantonese which is commonly used in academia and among Cantonese speakers. IPA and X-SAMPA are not commonly used but are available for English support. The IPA and X-SAMPA symbols in the table are for reference only and should not be used for Chinese transcription. Jyutping examples and the corresponding visemes are also shown.

To make Amazon Polly use phonetic pronunciation with Jyutping, use the phoneme alphabet="x-amazon-*jyutping*" tag.

The following examples show this with each standard.

Jyutping:

```
< speak >
  ## < phoneme alphabet="x-amazon-jyutping" ph="sing2">#</phoneme>#
  ## < phoneme alphabet="x-amazon-jyutping" ph="seng2">#</phoneme>#
</ speak >
```

IPA:

```
< speak >
  ## < phoneme alphabet="ipa" ph="p##k##n">pecan</phoneme>#
  ## < phoneme alphabet="ipa" ph="#pi.kæn">pecan</phoneme>#
</ speak >
```

X-SAMPA:

```
< speak >
```

```
## <phoneme alphabet='x-sampa' ph='pI"KA:n'>pecan</phoneme>#
## <phoneme alphabet='x-sampa' ph='"pi.k{n'>pecan</phoneme>#
</speak>
```

Note

Amazon Polly accepts Cantonese input encoded in UTF-8 only.

Phoneme/Viseme Table

Jyutping	IPA	X-SAMPA	Description	Jyutping Example	Viseme
Consonants					
b	p	p	voiceless bilabial plosive	巴, baa1	p
c	ts ^h	ts_h	aspirated voiceless alveolar affricate	叉, caa1	s
d	t	t	voiceless alveolar plosive	打, daa2	t
f	f	f	voiceless labiodental fricative	花, faa1	f
g	k	k	voiceless velar plosive	家, gaa1	k
gw	k ^w	k_w	labialized voiceless velar plosive	瓜, gwaa1	u
h	h	h	voiceless glottal fricative	哈, haa1	k
k	k ^h	k_h	aspirated voiceless velar plosive	卡, kaa1	k
kw	k ^{wh}	k_wh	labialized aspirated voiceless velar plosive	誇, kwaa1	u

Jyutping	IPA	X-SAMPA	Description	Jyutping Example	Viseme
l	l	l	alveolar lateral approximant	啦, laa1	t
m	m	m	bilabial nasal	媽, maa1	p
m	m	m=	syllabic bilabial nasal	唔, m4	p
ng	ŋ	N	velar nasal	牙, ngaa4	k
ng	ŋ	N=	syllabic velar nasal	吳, ng4	k
n	n	n	alveolar nasal	拿, naa4	t
p	p ^h	p_h	aspirated voiceless bilabial plosive	趴, paa1	p
s	s	s	voiceless alveolar fricative	沙, saa1	s
t	t ^h	t_h	aspirated voiceless alveolar plosive	他, taa1	t
w	w	w	labio-velar approximant	娃, waa1	u
y	j	j	palatal approximant	也, jaa5	i
z	ts	ts	voiceless alveolar affricate	渣, zaa1	s

Vowels

a	ɐ	6	near-open central vowel	吉, gat1	a
aa	ɑ	A	open back unrounded vowel	家, gaa1	a
aai	ai	Ai	diphthong	街, gaai1	a

Jyutping	IPA	X-SAMPA	Description	Jyutping Example	Viseme
aau	au	Au	diphthong	交, gaau1	a
ai	ɛi	6i	diphthong	雞, gai1	a
au	œu	6u	diphthong	溝, kau1	a
e	ɛ	E	open-mid front unrounded vowel	爹, de1	E
ei	ei	ei	diphthong	基, gei1	e
eo	ɵ	8	close-mid central rounded vowel	春, ceon1	o
eoɪ	øy	8y	diphthong	居, geoi1	o
eu	ɛu	Eu	diphthong	掉 in 掉垃圾, deu6	E
i	i	i	close front unrounded vowel	斯, si1	i
i	ɪ	ɪ	near-close near-front unrounded vowel	激, gik1	i
iu	iu	iu	diphthong	驕, giu1	i
o	ɔ	O	open-mid back rounded vowel	哥, go1	O
oe	œ	9	open-mid front rounded vowel	鋸, goe3	O
oi	ɔi	Oi	diphthong	該, goi1	O
ou	ou	ou	diphthong	高, gou1	o

Jyutping	IPA	X-SAMPA	Description	Jyutping Example	Viseme
u	u	u	close back rounded vowel	姑, gu1	u
u	ʊ	U	near-close near-back rounded vowel	谷, guk5	u
ui	ui	ui	diphthong	刼, gui6	u
yu	y	y	close front rounded vowel	於, jyu1	u
Tone marks and Additional Symbols					
1			high level	詩, si1	
2			medium rising	史, si2	
3			medium level	試, si3	
4			very low level	時, si4	
5			low rising	市, si5	
6			low level	是, si6	
-	.	.	syllable boundary	語音 jyu5-jam1	

Chinese (Mandarin) (cmn-CN)

The following table lists the Pinyin and International Phonetic Alphabet (IPA) phonemes for the Mandarin Chinese voice that is supported by Amazon Polly. Pinyin is the international standard for Standard Chinese romanization. IPA and X-SAMPA are not commonly used but are available for English support. The IPA and X-SAMPA symbols in the table are for reference only and should not be used for Chinese transcription. Pinyin examples and the corresponding visemes are also shown.

To make Amazon Polly use phonetic pronunciation with Pinyin, use the phoneme `alphabet="x-amazon-phonetic standard used"` tag.

The following examples show this with each standard.

Pinyin:

```
<speack>
  ## <phoneme alphabet="x-amazon-pinyin" ph="bo2">#</phoneme>#
  ## <phoneme alphabet="x-amazon-pinyin" ph="bao2">#</phoneme>#
</speack>
```

IPA:

```
<speack>
  ## <phoneme alphabet="ipa" ph="p##k##n">pecan</phoneme>#
  ## <phoneme alphabet="ipa" ph="#pi.kæn">pecan</phoneme>#
</speack>
```

X-SAMPA:

```
<speack>
  ## <phoneme alphabet='x-sampa' ph='pI"kA:n'>pecan</phoneme>#
  ## <phoneme alphabet='x-sampa' ph='"pi.k{n'>pecan</phoneme>#
</speack>
```

Note

Amazon Polly accepts Mandarin Chinese input encoded in UTF-8 only. The GB 18030 encoding standard is not currently supported by Amazon Polly.

Phoneme/Viseme Table

Pinyin	IPA	X-SAMPA	Description	Pinyin Example	Viseme
Consonants					

Pinyin	IPA	X-SAMPA	Description	Pinyin Example	Viseme
f	f	f	voiceless labiodental fricative	发, fa1	f
h	h	h	voiceless glottal fricative	和, he2	k
g	k	k	voiceless velar plosive	古, gu3	k
k	k ^h	k_h	aspirated voiceless velar plosive	苦, ku3	k
l	l	l	alveolar lateral approximant	拉, la1	t
m	m	m	bilabial nasal	骂, ma4	p
n	n	n	alveolar nasal	那, na4	t
ng	ŋ	N	velar nasal	正, zheng4	k
b	p	p	voiceless bilabial plosive	爸, ba4	p
p	p ^h	p_h	aspirated voiceless bilabial plosive	怕, pa4	p
s	s	s	voiceless alveolar fricative	四, si4	s
x	ç	s\	voiceless alveolo-palatal fricative	西, xi1	J
sh	ʂ	s`	voiceless retroflex fricative	是, shi4	S
d	t	t	voiceless alveolar plosive	打, da3	t

Pinyin	IPA	X-SAMPA	Description	Pinyin Example	Viseme
t	t ^h	t_h	aspirated voiceless alveolar plosive	他, ta1	t
zh	ʈʂ	t`s`	voiceless retroflex affricate	之, zhi1	S
ch	ʈʂ ^h	t`s`_h	aspirated voiceless retroflex affricate	吃, chi1	S
s	ʃ	ts	voiceless alveolar affricate	字, zi4	s
j	ʈʂ	ts\	voiceless alveolo-palatal affricate	鸡, ji1	J
q	ʈʂ ^h	ts_h	aspirated voiceless alveolo-palatal affricate	七, qi1	J
c	ʃ ^h	ts_h	aspirated voiceless alveolar affricate	次, ci4	s
w	w	w	labio-velar approximant	我, wo3	u
r	ʐ	z`	voiced retroflex fricative	日, ri4	S
"er" and "r" colored syllables					
er	ə	@`	r-coloured mid central vowel	二, er4	@
-r			r-colored syllable	馅儿, xianr4	@
Vowels					
e	ɤ	7	close-mid back unrounded vowel	恶, e4	e

Pinyin	IPA	X-SAMPA	Description	Pinyin Example	Viseme
e	ə	@	mid central vowel	恩, en1	@
a	a	a	open front unrounded vowel	安, an1	a
ai	aɪ	aɪ	diphthong	爱, ai4	a
ao	aʊ	aʊ	diphthong	奥, ao4	a
ei	eɪ	e	diphthong	诶, ei4	e
e	ɛ	E	open-mid front unrounded vowel	姐, jie3	E
i	i	i	close front unrounded vowel	鸡, ji1	i
ou	oʊ	oʊ	diphthong	欧, ou1	o
o	ɔ	O	open-mid back rounded vowel	哦, o4	o
u	u	u	close back rounded vowel	主, zhu3	u
yu	y	y	close front rounded vowel	于, yu2	u

Tone marks and Additional Symbols

1			high level tone	淤, yu1	
2			rising tone	鱼, yu2	
3			low (falling-rising) tone	语, yu3	
4			falling tone	育, yu4	

Pinyin	IPA	X-SAMPA	Description	Pinyin Example	Viseme
0			neutral tone	的, de0	
-	.	.	syllable boundary	语音 yu3-yin1	

Czech (cs-CZ)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Czech voice that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
p	p	voiceless bilabial plosive	pes	p
t	t	voiceless alveolar plosive	tok	t
c	c	voiceless palatal plosive	t'uk	J
k	k	voiceless velar plosive	kos	k
b	b	voiced bilabial plosive	bez	p
d	d	voiced alveolar plosive	dok	t
ɟ	J\	voiced palatal plosive	d'as	J

IPA	X-SAMPA	Description	Example	Viseme
g	g	voiced velar plosive	g um	k
f	f	voiceless labiodental fricative	f ilm	f
v	v	voiced labiodental fricative	v es	f
s	s	voiceless alveolar fricative	s en	s
z	z	voiced alveolar fricative	z el	s
ʃ	S	voiceless postalveolar fricative	š el	S
ʒ	Z	voiced postalveolar fricative	ž en	S
x	x	voiceless velar fricative	ch at	k
ħ	h	voiced glottal fricative	h us	k
ts	ts	voiceless alveolar affricate	c o	s
tʃ	tS	voiceless postalveolar affricate	č in	S
dz	dz	voiced alveolar affricate	š picberský	s
dʒ	dZ	voiceless alveolar affricate	dž in	S

IPA	X-SAMPA	Description	Example	Viseme
m	m	bilabial nasal	mor	p
n	n	alveolar nasal	nos	t
ɲ	J	palatal nasal	ňader	J
ŋ	N	velar nasal	banka	k
r	r	voiced alveolar trill	rys	r
ɾ	r_r	voiced raised alveolar fricative trill	řez	r
ɾ̥	r_0_r	voiceless raised alveolar fricative trill	keř	r
l	l	alveolar lateral approximant	les	t
j	j	palatal approximant	jen	i
w	w	labiovelar approximant	Watson	u
ɾ	r_ =	syllabic voiced alveolar trill	krk	r
ɹ̥	l_ =	syllabic alveolar lateral approximant	vl̥na	t
Vowels				
a	a	open front unrounded vowel	lan	a

IPA	X-SAMPA	Description	Example	Viseme
a:	a:	long open front unrounded vowel	lán	a
ɛ	E	open-mid front unrounded vowel	let	E
ɛ:	E:	long open-mid front unrounded vowel	lét	E
ɪ	ɪ	near-close near-front unrounded vowel	bit	i
i:	i:	long close front unrounded vowel	bít	i
o	o	close-mid back rounded vowel	hol	o
o:	o:	long close-mid back rounded vowel	gól	o
u	u	close back rounded vowel	pul	u
u:	u:	long close back rounded vowel	pŭl	u
au	au	diphthong	auto	a
ɛu	Eu	diphthong	euro	E
ou	ou	diphthong	mouk	o
Additional Symbols				

IPA	X-SAMPA	Description	Example	Viseme
'	"	primary stress		
.	.	syllable boundary		

Danish (da-DK)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Danish voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bat	p
d	d	voiced alveolar plosive	da	t
ð	D	voiced dental fricative	mad, thriller	T
f	f	voiceless labiodental fricative	fat	f
g	g	voiced velar plosive	gat	k
h	h	voiceless glottal fricative	hat	k
j	j	palatal approximant	jo	i

IPA	X-SAMPA	Description	Example	Viseme
k	k	voiceless velar plosive	kat	k
l	l	alveolar lateral approximant	ladt	t
m	m	bilabial nasal	mat	p
n	n	alveolar nasal	nay	t
ŋ	N	velar nasal	lang	k
p	p	voiceless bilabial plosive	pande	p
r	r	alveolar trill	thriller, story	r
ʀ	R	voiced uvular fricative	rat	k
s	s	voiceless alveolar fricative	sat	s
t	t	voiceless alveolar plosive	tal	t
v	v	voiced labiodental fricative	vat	f
w	w	labial-velar approximant	hav, weekend	u
Vowels				
ø	2	close-mid front rounded vowel	øst	o

IPA	X-SAMPA	Description	Example	Viseme
ø:	2:	long close-mid front rounded vowel	øse	o
ɐ	6	near-open central vowel	mor	a
œ	9	open-mid front rounded vowel	skøn, grønt	O
œ:	9:	long open-mid front rounded vowel	høne, gøre	O
ə	@	mid central vowel	ane	@
æ:	{:	long near-open front unrounded vowel	male	a
a	a	open front unrounded vowel	man	a
æ	{	near-open front unrounded vowel	adresse	a
ɑ	A	open back unrounded vowel	lak, tak	a
ɑ:	A:	long open back unrounded vowel	rase	a
e	e	close-mid front unrounded vowel	midt	e

IPA	X-SAMPA	Description	Example	Viseme
e:	e:	long close-mid front unrounded vowel	mele	e
ɛ	E	open-mid front unrounded vowel	mæt	E
ɛ:	E:	long open-mid front unrounded vowel	mæle	E
i	i	close front unrounded vowel	mit	i
i:	i:	long close front unrounded vowel	mile	i
o	o	close-mid back rounded vowel	foto	o
o:	o:	long close-mid back rounded vowel	mole	o
ɔ	O	open-mid back rounded vowel	mund	O
ɔ:	O:	long open-mid back rounded vowel	måle	O
ɒ:	Q:	long open back rounded vowel	morse	O
u	u	close back rounded vowel	lusk	u

IPA	X-SAMPA	Description	Example	Viseme
u:	u:	long close back rounded vowel	mule	u
ʌ	V	open-mid back unrounded	kører	E
y	y	close front rounded vowel	yt	u
y:	y:	long close front rounded vowel	hyle	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Dutch (Belgian) (nl-BE)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Belgian Dutch (Flemish) voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	b ak	p
d	d	voiced alveolar plosive	d ak	t

IPA	X-SAMPA	Description	Example	Viseme
ɖʒ	dʒ	voiced postalveolar affricate	manager	S
f	f	voiceless labiodental fricative	fel	f
ɡ	ɡ	voiced velar plosive	goal	k
ɣ	G	voiced velar fricative	hoed	k
ɦ	h\	voiced glottal fricative	hand	k
j	j	palatal approximant	ja	i
k	k	voiceless velar plosive	kap	k
l	l	alveolar lateral approximant	land	t
m	m	bilabial nasal	met	p
n	n	alveolar nasal	net	t
ŋ	N	velar nasal	bang	k
p	p	voiceless bilabial plosive	pak	p
r	r	alveolar trill	rand	r
s	s	voiceless alveolar fricative	sein	s

IPA	X-SAMPA	Description	Example	Viseme
ʃ	S	voiceless postalveolar fricative	show	S
t	t	voiceless alveolar plosive	tak	t
v	v	voiced labiodental fricative	vel	f
ʋ	v\	labiodental approximant	wit	f
x	x	voiceless velar fricative	toch	k
z	z	voiced alveolar fricative	ziin	s
ʒ	Z	voiced postalveolar fricative	bagage	S

Vowels

ø:	2:	long close-mid front rounded vowel	neus	o
œy	9y	diphthong	buit	O
ə	@	mid central vowel	de	@
a:	a:	long open front unrounded vowel	baad	a
ɑ:	A	open back unrounded vowel	bad	a

IPA	X-SAMPA	Description	Example	Viseme
e:	e:	long close-mid front unrounded vowel	beet	e
ɜ:	3:	long open-mid central unrounded vowel	barrière	E
ɛ	E	open-mid front unrounded vowel	bed	E
ɛi	Ei	diphthong	beet	E
i	i	close front unrounded vowel	vier	i
ɪ	ɪ	near-close near-front unrounded vowel	pit	i
o:	o:	long close-mid back rounded vowel	boot	o
ɔ	O	open-mid back rounded vowel	pot	O
u	u	close back rounded vowel	hoed	u
ʌu	Vu	diphthong	fout	E
y:	y:	long close front rounded vowel	fuut	u

IPA	X-SAMPA	Description	Example	Viseme
ʏ	Y	near-close near-front rounded vowel	hut	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Dutch (nl-NL)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Dutch voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	b ak	p
d	d	voiced alveolar plosive	d ak	t
ɖʒ	dʒ	voiced postalveolar affricate	man ag er	S
f	f	voiceless labiodental fricative	f el	f

IPA	X-SAMPA	Description	Example	Viseme
g	g	voiced velar plosive	goal	k
ɣ	G	voiced velar fricative	hoed	k
ɦ	h\	voiced glottal fricative	hand	k
j	j	palatal approximant	ja	i
k	k	voiceless velar plosive	kap	k
l	l	alveolar lateral approximant	land	t
m	m	bilabial nasal	met	p
n	n	alveolar nasal	net	t
ŋ	N	velar nasal	bang	k
p	p	voiceless bilabial plosive	pak	p
r	r	alveolar trill	rand	r
s	s	voiceless alveolar fricative	sein	s
ʃ	S	voiceless postalveolar fricative	show	S
t	t	voiceless alveolar plosive	tak	t

IPA	X-SAMPA	Description	Example	Viseme
v	v	voiced labiodental fricative	vel	f
ʋ	v\	labiodental approximant	wit	f
x	x	voiceless velar fricative	toch	k
z	z	voiced alveolar fricative	ziin	s
ʒ	Z	voiced postalveolar fricative	bagage	S

Vowels

ø:	2:	long close-mid front rounded vowel	neus	o
œy	9y	diphthong	buit	O
ə	@	mid central vowel	de	@
a:	a:	long open front unrounded vowel	baad	a
ɑ:	A	open back unrounded vowel	bad	a
e:	e:	long close-mid front unrounded vowel	beet	e
ɜ:	3:	long open-mid central unrounded vowel	barrière	E

IPA	X-SAMPA	Description	Example	Viseme
ɛ	E	open-mid front unrounded vowel	bed	E
ɛi	Ei	diphthong	beet	E
i	i	close front unrounded vowel	vier	i
ɪ	ɪ	near-close near-front unrounded vowel	pit	i
o:	o:	long close-mid back rounded vowel	boot	o
ɔ	O	open-mid back rounded vowel	pot	O
u	u	close back rounded vowel	hoed	u
ʌu	Vu	diphthong	fout	E
y:	y:	long close front rounded vowel	fuut	u
ʏ	Y	near-close near-front rounded vowel	hut	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

English (US) (en-US)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the American English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
dʒ	dZ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k

IPA	X-SAMPA	Description	Example	Viseme
l	l	alveolar lateral approximant	lay	l
m	m	bilabial nasal	mouse	p
n	n	alveolar nasal	nap	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	speak	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	trap	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u
z	z	voiced alveolar fricative	zero	s

IPA	X-SAMPA	Description	Example	Viseme
ʒ	Z	voiced postalveolar fricative	vision	S
Vowels				
ə	@	mid-central vowel	arena	@
ə̃	@`	mid-central r-colored vowel	reader	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ	A	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ̃	ʒ`	open mid-central unrounded r-colored vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
i	i	long close front unrounded vowel	fleece	i
ɪ	l	near-close near-front unrounded vowel	kit	i
oʊ	oU	diphthong	goat	o

IPA	X-SAMPA	Description	Example	Viseme
ɔ	O	long open mid-back rounded vowel	thought	O
ɔɪ	Oɪ	diphthong	choice	O
u	u	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʌ	V	open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

English (Australian) (en-AU)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Australian English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				

IPA	X-SAMPA	Description	Example	Viseme
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
ɖʒ	dʒ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k
l	l	alveolar lateral approximant	lay	t
ɭ	l=	syllabic alveolar lateral approximant	battle	t
m	m	bilabial nasal	mouse	p
ɱ	m=	syllabic bilabial nasal	anthem	p

IPA	X-SAMPA	Description	Example	Viseme
n	n	alveolar nasal	nap	t
ɳ	n=	syllabic alveolar nasal	button	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	pin	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	task	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u
z	z	voiced alveolar fricative	zero	s

IPA	X-SAMPA	Description	Example	Viseme
ʒ	Z	voiced postalveolar fricative	vision	S
Vowels				
ə	@	mid central vowel	arena	@
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ:	3:	long open mid-central unrounded vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E
i:	i	long close front unrounded vowel	fleece	i
ɪ	l	near-close near-front unrounded vowel	kit	i
ɪə	l@	diphthong	near	i

IPA	X-SAMPA	Description	Example	Viseme
ɔ:	OI	long open-mid back rounded vowel	thought	O
ɔɪ	OI	Diphthong	choice	O
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

English (British) (en-GB)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the British English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
dʒ	dʒ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k
l	l	alveolar lateral approximant	lay	t
l̩	l̩	syllabic alveolar lateral approximant	battle	t

IPA	X-SAMPA	Description	Example	Viseme
m	m	bilabial nasal	mouse	p
ṁ	m=	syllabic bilabial nasal	anthem	p
n	n	alveolar nasal	nap	t
ɳ	n=	syllabic alveolar nasal	button	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	pin	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	task	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u

IPA	X-SAMPA	Description	Example	Viseme
z	z	voiced alveolar fricative	zero	s
ʒ	Z	voiced postalveolar fricative	vision	S

Vowels

ə	@	mid central vowel	arena	@
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ:	3:	long open mid-central unrounded vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E
i:	i	long close front unrounded vowel	fleece	i

IPA	X-SAMPA	Description	Example	Viseme
ɪ	l	near-close near-front unrounded vowel	kit	i
ɪə	l@	diphthong	near	i
ɔ:	O:	long open-mid back rounded vowel	thought	O
ɔɪ	OI	Diphthong	choice	O
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

English (Indian) (en-IN)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Indian English voice supported by Amazon Polly.

For additional phonemes used in conjunction with Indian English, see [Hindi \(hi-IN\)](#).

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
ɖʒ	dʒ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k

IPA	X-SAMPA	Description	Example	Viseme
l	l	alveolar lateral approximant	lay	t
ɭ	l=	syllabic alveolar lateral approximant	battle	t
m	m	bilabial nasal	mouse	p
ṁ	m=	syllabic bilabial nasal	anthem	p
n	n	alveolar nasal	nap	t
ɳ	n=	syllabic alveolar nasal	nap	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	pin	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	task	t
ʧ	tS	voiceless postalveolar affricate	chart	S

IPA	X-SAMPA	Description	Example	Viseme
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u
z	z	voiced alveolar fricative	zero	s
ʒ	Z	voiced postalveolar fricative	vision	S

Vowels

ə	@	mid central vowel	arena	@
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ:	3:	long open mid-central unrounded vowel	nurse	E

IPA	X-SAMPA	Description	Example	Viseme
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E
i:	i	long close front unrounded vowel	fleece	i
ɪ	ɪ	near-close near-front unrounded vowel	kit	i
ɪə	ɪ@	diphthong	near	i
ɔ:	Oɪ	long open-mid back rounded vowel	thought	O
ɔɪ	Oɪ	Diphthong	choice	O
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	

IPA	X-SAMPA	Description	Example	Viseme
,	%	secondary stress	Alabama	
.	.	syllable boundary	A.la.ba.ma	

English (Ireland) (en-IE)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Irish English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
ɟʒ	dʒ	voiced postalveolar affricate	jump	s
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k

IPA	X-SAMPA	Description	Example	Viseme
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k
l	l	alveolar lateral approximant	lay	t
m	m	bilabial nasal	mouse	p
n	n	alveolar nasal	nap	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	speak	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	trap	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f

IPA	X-SAMPA	Description	Example	Viseme
w	w	labial-velar approximant	west	u
z	z	voiced alveolar fricative	zero	s
ʒ	Z	voiced postalveolar fricative	vision	S
Vowels				
ə	@	mid-central vowel	arena	@
ə̃	@`	mid-central r-colored vowel	reader	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ	A	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ̃	ʒ`	open mid-central unrounded r-colored vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
i	i	long close front unrounded vowel	fleece	i

IPA	X-SAMPA	Description	Example	Viseme
ɪ	ɪ	near-close near-front unrounded vowel	kit	i
oʊ	oU	diphthong	goat	o
ɔ	O	long open mid-back rounded vowel	thought	O
ɔɪ	Oɪ	diphthong	choice	O
u	u	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʌ	V	open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

English (New Zealand) (en-NZ)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the New Zealand English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
dʒ	dʒ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k
l	l	alveolar lateral approximant	lay	t
l̩	l̩	syllabic alveolar lateral approximant	battle	t

IPA	X-SAMPA	Description	Example	Viseme
m	m	bilabial nasal	mouse	p
ṁ	m=	syllabic bilabial nasal	anthem	p
n	n	alveolar nasal	nap	t
ɳ	n=	syllabic alveolar nasal	button	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	pin	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	task	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u

IPA	X-SAMPA	Description	Example	Viseme
z	z	voiced alveolar fricative	zero	s
ʒ	Z	voiced postalveolar fricative	vision	S

Vowels

ə	@	mid central vowel	arena	@
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ:	3:	long open mid-central unrounded vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E
i:	i	long close front unrounded vowel	fleece	i

IPA	X-SAMPA	Description	Example	Viseme
ɪ	ɪ	near-close near-front unrounded vowel	kit	i
ɪə	ɪ@	diphthong	near	i
ɔ:	O:	long open-mid back rounded vowel	thought	O
ɔɪ	Oɪ	Diphthong	choice	O
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

The Aria voice speaks New Zealand English and offers limited support for Maori. It can pronounce the following Maori words and phrases. The Maori phrases are case-sensitive.

English	Maori
Hello/cheers	Kia ora
Welcome (to)	Nau mai (ki)
Hello (one person)/thank you	Tēnā koe
Hello (three or more people)/thank you	Tēnā koutou
Good morning	Ata mārie
Good morning	Mōrena
Thank you	Ngā mihi
Take care	Ngā manaakitanga
See you	Ka kite
See you later	Mā te wā
Have a good day	Kia pai tō rā
Merry Christmas	Meri Kirihimete
Maori	Māori
Maori language	te reo Māori
Maori language week	Te wiki o te reo Māori
New Zealand	Aotearoa
Maori New Year	Mātariki
Town in New Zealand / Waitangi Day is the national day of New Zealand	Waitangi
One	tahi
Two	rua

English	Maori
Three	toru
Four	whā
Five	rima
Six	ono
Seven	whitu
Eight	waru
Nine	iwa
Ten	tekau
Twenty	rua tekau
Thirty	Toru tekau

English (Singaporean) (en-SG)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Singaporean English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t

IPA	X-SAMPA	Description	Example	Viseme
ɹ̥ʒ	dZ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k
l	l	alveolar lateral approximant	lay	t
ɭ	l=	syllabic alveolar lateral approximant	battle	t
m	m	bilabial nasal	mouse	p
ᵿ	m=	syllabic bilabial nasal	anthem	p
n	n	alveolar nasal	nap	t
ɳ	n=	syllabic alveolar nasal	button	t

IPA	X-SAMPA	Description	Example	Viseme
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	pin	p
ɹ	r\	alveolar approximant	red	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	task	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u
z	z	voiced alveolar fricative	zero	s
ʒ	Z	voiced postalveolar fricative	vision	S
Vowels				
ə	@	mid central vowel	arena	@

IPA	X-SAMPA	Description	Example	Viseme
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ:	3:	long open mid-central unrounded vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E
i:	i	long close front unrounded vowel	fleece	i
ɪ	l	near-close near-front unrounded vowel	kit	i
ɪə	l@	diphthong	near	i
ɔ:	O:	long open-mid back rounded vowel	thought	O
ɔɪ	Oɪ	Diphthong	choice	O

IPA	X-SAMPA	Description	Example	Viseme
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

English (South African) (en-ZA)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the South African English voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p

IPA	X-SAMPA	Description	Example	Viseme
d	d	voiced alveolar plosive	dig	t
ɖʒ	dʒ	voiced postalveolar affricate	jump	S
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k
j	j	palatal approximant	yes	i
k	k	voiceless velar plosive	cat	k
l	l	alveolar lateral approximant	lay	t
ɭ	l=	syllabic alveolar lateral approximant	battle	t
ɬ	K	voiceless lateral fricative	umhlanga	t
m	m	bilabial nasal	mouse	p
ᵹ	m=	syllabic bilabial nasal	anthem	p

IPA	X-SAMPA	Description	Example	Viseme
n	n	alveolar nasal	nap	t
ɳ	n=	syllabic alveolar nasal	button	t
ŋ	N	velar nasal	thing	k
p	p	voiceless bilabial plosive	pin	p
ɹ	r\	alveolar approximant	red	r
r	r	alveolar trill	pareis	r
s	s	voiceless alveolar fricative	seem	s
ʃ	S	voiceless postalveolar fricative	ship	S
t	t	voiceless alveolar plosive	task	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u
x	x	voiceless velar fricative	gauteng	k

IPA	X-SAMPA	Description	Example	Viseme
z	z	voiced alveolar fricative	zero	s
!	!\	post-alveolar click	gqeberha	k
	\	dental click	ncube	t
	\	lateral click	xhosa	t
Vowels				
ə	@	mid central vowel	arena	@
əi	@i	diphthong	nelspruit	i
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e
ɜ:	3:	long open mid-central unrounded vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E

IPA	X-SAMPA	Description	Example	Viseme
i:	i	long close front unrounded vowel	fleece	i
ɪə	l@	diphthong	du preez	i
ɪ	l	near-close near-front unrounded vowel	kit	i
ɪə	l@	diphthong	near	i
ɔ:	O:	long open-mid back rounded vowel	thought	O
ɔɪ	Oɪ	Diphthong	choice	O
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E
y	y	close front rounded vowel	van vuuren	u
Additional Symbols				
'	"	primary stress	Alab a ma	

IPA	X-SAMPA	Description	Example	Viseme
,	%	secondary stress	Alabama	
.	.	syllable boundary	A.la.ba.ma	

English (Welsh) (en-GB-WLS)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Welsh English voice supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bed	p
d	d	voiced alveolar plosive	dig	t
ɖʒ	dʒ	voiced postalveolar affricate	jump	s
ð	D	voiced dental fricative	then	T
f	f	voiceless labiodental fricative	five	f
g	g	voiced velar plosive	game	k
h	h	voiceless glottal fricative	house	k

IPA	X-SAMPA	Description	Example	Viseme
j	j	palatal approximant	y es	i
k	k	voiceless velar plosive	c at	k
l	l	alveolar lateral approximant	l ay	t
ɭ	l=	syllabic alveolar lateral approximant	ba ttle	t
m	m	bilabial nasal	m ouse	p
ᵿ	m=	syllabic bilabial nasal	an th m	p
n	n	alveolar nasal	n ap	t
ɳ	n=	syllabic alveolar nasal	na p	t
ŋ	N	velar nasal	thi ng	k
p	p	voiceless bilabial plosive	p in	p
ɹ	r\	alveolar approximant	r ed	r
s	s	voiceless alveolar fricative	s een	s
ʃ	S	voiceless postalveolar fricative	sh ip	S

IPA	X-SAMPA	Description	Example	Viseme
t	t	voiceless alveolar plosive	task	t
tʃ	tS	voiceless postalveolar affricate	chart	S
θ	T	voiceless dental fricative	thin	T
v	v	voiced labiodental fricative	vest	f
w	w	labial-velar approximant	west	u
z	z	voiced alveolar fricative	zero	s
ʒ	Z	voiced postalveolar fricative	vision	S

Vowels

ə	@	mid central vowel	arena	@
əʊ	@U	diphthong	goat	@
æ	{	near open-front unrounded vowel	trap	a
aɪ	al	diphthong	price	a
aʊ	aU	diphthong	mouth	a
ɑ:	A:	long open-back unrounded vowel	father	a
eɪ	el	diphthong	face	e

IPA	X-SAMPA	Description	Example	Viseme
ɜ:	3:	long open mid-central unrounded vowel	nurse	E
ɛ	E	open mid-front unrounded vowel	dress	E
ɛə	E@	diphthong	square	E
i:	i	long close front unrounded vowel	fleece	i
ɪ	I	near-close near-front unrounded vowel	kit	i
ɪə	I@	diphthong	near	i
ɔ:	OI	long open-mid back rounded vowel	thought	O
ɔɪ	OI	Diphthong	choice	O
ɒ	Q	open back rounded vowel	lot	O
u:	u:	long close-back rounded vowel	goose	u
ʊ	U	near-close near-back rounded vowel	foot	u
ʊə	U@	diphthong	cure	u
ʌ	V	Open-mid-back unrounded vowel	strut	E

IPA	X-SAMPA	Description	Example	Viseme
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Finnish (fi-FI)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Finnish voice that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Finnish consonants				
p	p	voiceless bilabial plosive	[p]ankki	p
t	t	voiceless alveolar plosive	[t]alo	t
k	k	voiceless velar plosive	[k]aali	k
d	d	voiced alveolar plosive	[d]ata	t
s	s	voiceless alveolar fricative	[s]ali	s
h	h	voiceless glottal fricative	[h]attu	k

IPA	X-SAMPA	Description	Example	Viseme
ʊ	v\	voiced labiodental approximant	[v]aiṽā	v
j	j	palatal approximant	[j]oki	i
l	l	alveolar lateral approximant	[l]oma	t
r	r	voiced alveolar trill	[r]iita	r
m	m	bilabial nasal	[m]ato	p
n	n	alveolar nasal	[n]enää	t
ŋ	N	velar nasal	he[n]ki	k

Consonants found in loanwords

b	b	voiced bilabial plosive	[b]ussi	p
f	f	voiceless labiodental fricative	[f]irma	v
w	w	labial-velar approximant	[w]iki	u
z	z	voiced alveolar fricative	[z]ulu	s
g	g	voiced velar plosive	[g]aala	k
ʃ	S	voiceless postalveolar fricative	[sh]akki	S

IPA	X-SAMPA	Description	Example	Viseme
ʒ	Z	voiced postalveolar fricative	[g]enre	S
θ	T	voiceless dental fricative	ear[th]	T
ð	D	voiced dental fricative	ei[th]er	T

Short vowels

i	i	close front unrounded vowel	k[i]lo	i
ɛ	E	open mid-front unrounded vowel	k[e]sä	E
æ	{	near open-front unrounded vowel	k[ä]ly	A
y	y	close front rounded vowel	k[y]lä	u
ø	2	close mid-front rounded vowel	p[ö]ly	O
u	u	close back rounded vowel	k[u]lo	u
ɔ	O	open mid-back rounded vowel	k[o]lo	O
ɑ	A	open back unrounded vowel	k[a]la	A

Long vowels

IPA	X-SAMPA	Description	Example	Viseme
i:	i:	long close front unrounded vowel	s[ii]li	i
ɛ:	E:	long open mid-front unrounded vowel	[ee]tu	E
æ:	{:	long near open-front unrounded vowel	t[ää]llä	A
y:	y:	long close front unrounded vowel	t[yy]li	u
ø:	2:	long close mid-front rounded vowel	t[öö]lö	O
u:	u:	long close back rounded vowel	t[uu]li	u
ɔ:	O:	long open mid-back rounded vowel	r[oo]li	O
ɑ:	A:	long open back unrounded vowel	k[aa]su	A

Diphthongs

ɛi	Ei	diphthong	l[ei]pä	E
æi	{i	diphthong	[äi]ti	A
ui	ui	diphthong	k[ui]n	u
ai	Ai	diphthong	k[ai]kki	A

IPA	X-SAMPA	Description	Example	Viseme
ɔi	Oi	diphthong	p[oi]ka	O
øi	2i	diphthong	s[öi]n	O
yi	yi	diphthong	l[yi]jy	u
au	Au	diphthong	s[au]na	A
ɔu	Ou	diphthong	k[ou]lu	O
ɛu	Eu	diphthong	r[eu]na	E
iu	iu	diphthong	v[iu]lu	i
æy	{y	diphthong	t[äy]nnä	A
øy	2y	diphthong	k[öy]hä	O
ɛy	Ey	diphthong	pes[ey]tyä	E
iy	iy	diphthong	käär[iy]tyä	i
iɛ	iE	diphthong	t[ie]	i
yø	y2	diphthong	[yö]	u
uo	uO	diphthong	t[uo]	u
Vowels found in English loanwords				
ɪ	I	near-close near-front unrounded vowel	b[i]t	i
ʊ	U	near-close near-back rounded vowel	b[oo]k	u
ə	@	mid-central vowel	[a]bout	@

IPA	X-SAMPA	Description	Example	Viseme
ʌ	V	open-mid-back unrounded vowel	c[u]t	E

French (fr-FR)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the French voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bo ire	p
d	d	voiced alveolar plosive	ma d ame	t
f	f	voiceless labiodental fricative	f emme	f
g	g	voiced velar plosive	g rand	k
ɥ	H	labial-palatal approximant	br uit	u
j	j	palatal approximant	me ill eur	i
k	k	voiceless velar plosive	q atre	k

IPA	X-SAMPA	Description	Example	Viseme
l	l	alveolar lateral approximant	malade	t
m	m	bilabial nasal	maison	p
n	n	alveolar nasal	astronome	t
ɲ	J	palatal nasal	baigner	J
ŋ	N	velar nasal	parking	k
p	p	voiceless bilabial plosive	pomme	p
ʁ	R	voiced uvular fricative	amoureux	k
s	s	voiceless alveolar fricative	santé	s
ʃ	S	voiceless postalveolar fricative	chat	S
t	t	voiceless alveolar plosive	téléphone	t
v	v	voiced labiodental fricative	vrai	f
w	w	labial-velar approximant	soir	u
z	z	voiced alveolar fricative	raison	s
ʒ	Z	voiced postalveolar fricative	aubergine	S

IPA	X-SAMPA	Description	Example	Viseme
Vowels				
ø	2	close-mid front rounded vowel	deux	o
œ	9	open-mid front rounded vowel	neuf	O
œ̃	9~	nasal open-mid front rounded vowel	brun	O
ə	@	mid central vowel	je	@
a	a	open front unrounded vowel	table	a
ã	A~	nasal open back unrounded vowel	camembert	a
e	e	close-mid front unrounded vowel	marché	e
ɛ	E	open-mid front unrounded vowel	neige	E
ẽ	E~	nasal open-mid front unrounded vowel	sapin	E
i	i	close front unrounded vowel	mille	i
o	o	close-mid back rounded vowel	hôpital	o
ɔ	O	open-mid back rounded vowel	homme	O

IPA	X-SAMPA	Description	Example	Viseme
õ	O~	nasal open-mid back rounded vowel	bon	O
u	u	close back rounded vowel	sous	u
y	y	close front rounded vowel	dur	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

French (Belgian) (fr-BE)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Belgian French voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	boire	p
d	d	voiced alveolar plosive	madame	t

IPA	X-SAMPA	Description	Example	Viseme
f	f	voiceless labiodental fricative	femme	f
g	g	voiced velar plosive	grand	k
ɥ	H	labial-palatal approximant	bruit	u
j	j	palatal approximant	meilleur	i
k	k	voiceless velar plosive	quatre	k
l	l	alveolar lateral approximant	malade	t
m	m	bilabial nasal	maison	p
n	n	alveolar nasal	astronome	t
ɲ	J	palatal nasal	baigner	J
ŋ	N	velar nasal	parking	k
p	p	voiceless bilabial plosive	pomme	p
ʁ	R	voiced uvular fricative	amoureux	k
s	s	voiceless alveolar fricative	santé	s
ʃ	S	voiceless postalveolar fricative	chat	S

IPA	X-SAMPA	Description	Example	Viseme
t	t	voiceless alveolar plosive	téléphone	t
v	v	voiced labiodental fricative	vrai	f
w	w	labial-velar approximant	soir	u
z	z	voiced alveolar fricative	raison	s
ʒ	Z	voiced postalveolar fricative	aubergine	S

Vowels

ø	2	close-mid front rounded vowel	deux	o
œ	9	open-mid front rounded vowel	neuf	O
œ̃	9~	nasal open-mid front rounded vowel	brun	O
ə	@	mid central vowel	je	@
a	a	open front unrounded vowel	table	a
ã	A~	nasal open back unrounded vowel	camembert	a
e	e	close-mid front unrounded vowel	marché	e

IPA	X-SAMPA	Description	Example	Viseme
ɛ	E	open-mid front unrounded vowel	neige	E
ẽ	E~	nasal open-mid front unrounded vowel	sapin	E
i	i	close front unrounded vowel	mille	i
o	o	close-mid back rounded vowel	hôpital	o
ɔ	O	open-mid back rounded vowel	homme	O
õ	O~	nasal open-mid back rounded vowel	bon	O
u	u	close back rounded vowel	sous	u
y	y	close front rounded vowel	dur	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

French (Canadian) (fr-CA)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the French Canadian voice supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bo ire	p
d	d	voiced alveolar plosive	ma d ame	t
f	f	voiceless labiodental fricative	f emme	f
g	g	voiced velar plosive	g rand	k
ɥ	H	labial-palatal approximant	bru it	u
j	j	palatal approximant	me ill eur	i
k	k	voiceless velar plosive	q atre	k
l	l	alveolar lateral approximant	ma l ade	t
m	m	bilabial nasal	m aison	p
n	n	alveolar nasal	astron o me	t
ɲ	J	palatal nasal	ba igner	J

IPA	X-SAMPA	Description	Example	Viseme
ŋ	N	velar nasal	parking	k
p	p	voiceless bilabial plosive	pomme	p
ʁ	R	voiced uvular fricative	amoureux	k
s	s	voiceless alveolar fricative	sant�	s
ʃ	S	voiceless postalveolar fricative	chat	S
t	t	voiceless alveolar plosive	t�l�phone	t
v	v	voiced labiodental fricative	vrai	f
w	w	labial-velar approximant	soir	u
z	z	voiced alveolar fricative	raison	s
ʒ	Z	voiced postalveolar fricative	aubergine	S

Vowels

�	2	close-mid front rounded vowel	deux	o
�	9	open-mid front rounded vowel	neuf	O

IPA	X-SAMPA	Description	Example	Viseme
œ̃	9~	nasal open-mid front rounded vowel	brun	O
ə	@	mid central vowel	je	@
a	a	open front unrounded vowel	table	a
ã	A~	nasal open back unrounded vowel	camembert	a
e	e	close-mid front unrounded vowel	marché	e
ɛ	E	open-mid front unrounded vowel	neige	E
ẽ	E~	nasal open-mid front unrounded vowel	sapin	E
i	i	close front unrounded vowel	mille	i
o	o	close-mid back rounded vowel	hôpital	o
ɔ	O	open-mid back rounded vowel	homme	O
õ	O~	nasal open-mid back rounded vowel	bon	O
u	u	close back rounded vowel	sous	u

IPA	X-SAMPA	Description	Example	Viseme
y	y	close front rounded vowel	dur	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

German (de-DE)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the German voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ʔ	ʔ	glottal stop		
b	b	voiced bilabial plosive	B ier	p
d	d	voiced alveolar plosive	D ach	t
ç	C	voiceless palatal fricative	i ch	k
ɟ͡ʒ	dZ	voiced postalveolar affricate	D schungel	S

IPA	X-SAMPA	Description	Example	Viseme
f	f	Voiceless labiodental fricative	V ogel	f
g	g	Voiced velar plosive	G abel	k
h	h	Voiceless glottal fricative	H aus	k
j	j	Voiceless glottal fricative	j emand	i
k	k	Voiceless velar plosive	K leid	k
l	l	Alveolar lateral approximant	L och	t
m	m	Bilabial nasal	M ilch	p
n	n	Alveolar nasal	N atur	t
ŋ	N	Velar nasal	k lingen	k
p	p	Voiceless bilabial plosive	P ark	p
pf	pf	Voiceless labiodental affricate	A pfel	
R	R	Uvular trill	R egen	
s	s	voiceless alveolar fricative	M esser	s
ʃ	S	Voiceless postalveolar fricative	F ischer	S

IPA	X-SAMPA	Description	Example	Viseme
t	t	Voiceless alveolar plosive	Topf	T
ts	Ts	Voiceless alveolar affricate	Zahl	
tʃ	tS	Voiceless postalveolar affricate	deutsch	S
v	v	Voiced labiodental fricative	Wasser	f
x	x	Voiceless velar fricative	kochen	k
z	z	Voiced alveolar fricative	See	s
ʒ	Z	Voiced postalveolar fricative	Orange	S
Vowels				
øː	2ː	long close-mid front rounded vowel	böse	o
ə	6	near-open central vowel	besser	a
æ̯	6_^	non-syllabic near-open central vowel	Klar	a
œ	9	open-mid front rounded vowel	können	O
ə	@	mid central vowel	Rede	@

IPA	X-SAMPA	Description	Example	Viseme
a	a	open front unrounded vowel	Salz	a
a:	a:	long open front unrounded vowel	Sahne	a
aɪ	aɪ	diphthong	nein	a
aʊ	aʊ	diphthong	Augen	a
ã	A~	nasal open back unrounded vowel	Restaurant	a
e:	e:	long close-mid front unrounded vowel	Rede	e
ɛ	E	open-mid front unrounded vowel	Keller	E
ẽ	E~	nasal open-mid front unrounded vowel	Terrain	E
i:	i:	long close front unrounded vowel	Lied	i
ɪ	ɪ	near-close near-front unrounded vowel	bitte	i
o:	o:	long close-mid back rounded vowel	Kohl	o
ɔ	O	open-mid back rounded vowel	Koffer	O

IPA	X-SAMPA	Description	Example	Viseme
õ	O~	nasal open-mid back rounded vowel	Annonce	O
ɔʏ	OY	diphthong	neu	O
u:	u:	long close back rounded vowel	Bruder	u
ʊ	U	near-close near-back rounded vowel	Wunder	u
y:	y:	long close front rounded vowel	kühl	u
ʏ	Y	near-close near-front rounded vowel	Küche	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

German (Austrian) (de-AT)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Austrian German voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ʔ	ʔ	glottal stop		
b	b	voiced bilabial plosive	B ier	p
d	d	voiced alveolar plosive	D ach	t
ç	C	voiceless palatal fricative	i ch	k
ɟʒ	dZ	voiced postalveolar affricate	D schungel	s
f	f	Voiceless labiodental fricative	V ogel	f
g	g	Voiced velar plosive	G abel	k
h	h	Voiceless glottal fricative	H aus	k
j	j	Voiceless glottal fricative	j emand	i
k	k	Voiceless velar plosive	K leid	k
l	l	Alveolar lateral approximant	L och	t
m	m	Bilabial nasal	M ilch	p
n	n	Alveolar nasal	N atur	t

IPA	X-SAMPA	Description	Example	Viseme
ŋ	N	Velar nasal	klingen	k
p	p	Voiceless bilabial plosive	Park	p
pf	pf	Voiceless labiodental affricate	Apfel	
R	R	Uvular trill	Regen	
s	s	voiceless alveolar fricative	Messer	s
ʃ	S	Voiceless postalveolar fricative	Fischer	S
t	t	Voiceless alveolar plosive	Topf	T
ts	Ts	Voiceless alveolar affricate	Zahl	
tʃ	tS	Voiceless postalveolar affricate	deutsch	S
v	v	Voiced labiodental fricative	Wasser	f
x	x	Voiceless velar fricative	kochen	k
z	z	Voiced alveolar fricative	See	s
ʒ	Z	Voiced postalveolar fricative	Orange	S

IPA	X-SAMPA	Description	Example	Viseme
Vowels				
ø:	2:	long close-mid front rounded vowel	böse	o
ɐ	6	near-open central vowel	besser	a
ɐ̯	6_^	non-syllabic near-open central vowel	Klar	a
œ	9	open-mid front rounded vowel	können	O
ə	@	mid central vowel	Rede	@
a	a	open front unrounded vowel	Salz	a
a:	a:	long open front unrounded vowel	Sahne	a
aɪ	al	diphthong	nein	a
aʊ	aU	diphthong	Augen	a
ã	A~	nasal open back unrounded vowel	Restaurant	a
e:	e:	long close-mid front unrounded vowel	Rede	e
ɛ	E	open-mid front unrounded vowel	Keller	E

IPA	X-SAMPA	Description	Example	Viseme
ɛ̃	E~	nasal open-mid front unrounded vowel	Terrain	E
i:	i:	long close front unrounded vowel	Lied	i
ɪ	ɪ	near-close near-front unrounded vowel	bitte	i
o:	o:	long close-mid back rounded vowel	Kohl	o
ɔ	O	open-mid back rounded vowel	Koffer	O
õ	O~	nasal open-mid back rounded vowel	Annonce	O
ɔʏ	OY	diphthong	neu	O
u:	u:	long close back rounded vowel	Bruder	u
ʊ	U	near-close near-back rounded vowel	Wunder	u
y:	y:	long close front rounded vowel	kühl	u
ʏ	Y	near-close near-front rounded vowel	Küche	u

IPA	X-SAMPA	Description	Example	Viseme
Additional Symbols				
'	"	primary stress	Al a bama	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

German (Swiss standard) (de-CH)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the German (Swiss standard) voice that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ʔ	ʔ	glottal stop		
b	b	voiced bilabial plosive	B ier	p
d	d	voiced alveolar plosive	D ach	t
ç	C	voiceless palatal fricative	i ch	k
ɟʒ	dZ	voiced postalveolar affricate	D schungel	S
f	f	Voiceless labiodental fricative	V ogel	f

IPA	X-SAMPA	Description	Example	Viseme
g	g	Voiced velar plosive	G abel	k
h	h	Voiceless glottal fricative	H aus	k
j	j	Voiceless glottal fricative	j emand	i
k	k	Voiceless velar plosive	K leid	k
l	l	Alveolar lateral approximant	L och	t
m	m	Bilabial nasal	M ilch	p
n	n	Alveolar nasal	N atur	t
ŋ	N	Velar nasal	k lingen	k
p	p	Voiceless bilabial plosive	P ark	p
pf	pf	Voiceless labiodental affricate	A pfel	
R	R	Uvular trill	R egen	
s	s	voiceless alveolar fricative	M esser	s
ʃ	S	Voiceless postalveolar fricative	F ischer	S
t	t	Voiceless alveolar plosive	T opf	T

IPA	X-SAMPA	Description	Example	Viseme
ts	Ts	Voiceless alveolar affricate	Zahl	
tʃ	tS	Voiceless postalveolar affricate	deutsch	S
v	v	Voiced labiodental fricative	Wasser	f
x	x	Voiceless velar fricative	kochen	k
z	z	Voiced alveolar fricative	See	s
ʒ	Z	Voiced postalveolar fricative	Orange	S
Vowels				
ø:	2:	long close-mid front rounded vowel	böse	o
ɐ	6	near-open central vowel	besser	a
æ̯	6_^	non-syllabic near-open central vowel	Klar	a
œ	9	open-mid front rounded vowel	können	O
ə	@	mid central vowel	Rede	@
a	a	open front unrounded vowel	Salz	a

IPA	X-SAMPA	Description	Example	Viseme
a:	a:	long open front unrounded vowel	Sahne	a
aɪ	aɪ	diphthong	nein	a
aʊ	aʊ	diphthong	Augen	a
ã	A~	nasal open back unrounded vowel	Restaurant	a
e:	e:	long close-mid front unrounded vowel	Rede	e
ɛ	E	open-mid front unrounded vowel	Keller	E
ẽ	E~	nasal open-mid front unrounded vowel	Terrain	E
i:	i:	long close front unrounded vowel	Lied	i
ɪ	ɪ	near-close near-front unrounded vowel	bitte	i
o:	o:	long close-mid back rounded vowel	Kohl	o
ɔ	O	open-mid back rounded vowel	Koffer	O

IPA	X-SAMPA	Description	Example	Viseme
õ	O~	nasal open-mid back rounded vowel	Annonce	O
ɔʏ	OY	diphthong	neu	O
u:	u:	long close back rounded vowel	Bruder	u
ʊ	U	near-close near-back rounded vowel	Wunder	u
y:	y:	long close front rounded vowel	kühl	u
ʏ	Y	near-close near-front rounded vowel	Küche	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Hindi (hi-IN)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the phoneme's sound type for the Hindi voices that are supported by Amazon Polly.

For additional phonemes used in conjunction with Hindi, see [English \(Indian\) \(en-IN\)](#).

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example
Consonants			
p ^h	p_h	voiceless aspirated bilabial plosive	फूल (phool)
b ^h	b_h	voiced aspirated bilabial plosive	भारी (bhaari)
t̪	t_d	voiceless dental plosive	तापमान (taapmaan)
t̪ ^h	t_d_h	voiceless aspirated dental plosive	थोड़ा (thoda)
d̪	d_d	voiced dental plosive	दिल्ली (dilli)
d̪ ^h	d_d_h	voiced aspirated dental plosive	धोबी (dhobi)
t̪	t`	voiceless retroflex plosive	कटोरा (katora)
t̪ ^h	t`_h	voiceless aspirated retroflex plosive	ठंड (thand)
d̪	d`	voiced retroflex plosive	डर (darr)
d̪ ^h	d`_h	voiced aspirated retroflex plosive	ढाल (dhal)
tʃ ^h	tS_h	voiceless aspirated palatal affricate	छाल (chaal)
dʒ ^h	dZ_h	voiced aspirated palatal affricate	झाल (jhaal)
k ^h	k_h	voiceless aspirated velar plosive	खान (khan)

IPA	X-SAMPA	Description	Example
g ^h	g_h	voiced aspirated velar plosive	घान (ghaan)
ŋ	n`	retroflex nasal	क्षण (kshan)
r	4	alveolar flap	राम (ram)
ɽ	r`	plain retroflex flap	बड़ा (bada)
ɽ ^h	r`_h	voiced aspirated retroflex flap	बढ़ी (barhi)
ʋ	v\	bilabial approximant	वसूल (wasool)
Vowels			
ə	@_o	mid central vowel	अच्छा (achhaa)
ẽ	@~	nasalised mid central vowel	हँसना (hansnaa)
a	A_o	open front unrounded vowel	आग (aag)
a~	A~	nasalised open front unrounded vowel	घड़ियाँ (ghariyaan)
ɪ	l_o	near-close near-front unrounded vowel	इक्कीस (ikkees)
ĩ	l~	nasalised near-close near front unrounded vowel	संचिर्ई (sinchai)
i	i_o	close front unrounded vowel	बिल्ली (billee)
ĩ	i~	nasalised close front unrounded vowel	नहीं (nahin)

IPA	X-SAMPA	Description	Example
ʊ	U_o	near-close near-back rounded vowel	उलूल (ullu)
ʊ̃	U~	nasalised near-close near-back rounded vowel	मुँह (munh)
u	u_o	close back rounded vowel	फूल (phool)
u~	u~	nasalised close back rounded vowel	ऊँट (oont)
ɔ	O_o	open-mid back rounded vowel	कौन (kaun)
ɔ̃	O~	nasalised open-mid back rounded vowel	भौ (bhaun)
o	o	close-mid back rounded vowel	सोना (sona)
o~	o~	nasalised close-mid back rounded vowel	क्यो (kyon)
ɛ	E_o	open-mid front unrounded vowel	पैसा (paisa)
ɛ̃	E~	nasalised open-mid front unrounded vowel	मैं (main)
e	e	close-mid front unrounded vowel	एक (ek)
e~	e~	nasalised close-mid front unrounded vowel	कतिबे (kitabein)

Icelandic (is-IS)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Icelandic voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	gras b akkanum	0
c	c	voiceless palatal plosive	pa k kin	k
c ^h	c_h	aspirated voiceless palatal plosive	anarkista i	k
ç	C	voiceless palatal fricative	h e ðan	k
d	d	voiced alveolar plosive	bón d i	t
ð	D	voiced dental fricative	bor ð	T
f	f	voiceless labiodental fricative	du f t	f
g	g	voiced velar plosive	holg ó ma	k
ɣ	G	voiced velar fricative	hug u r	k

IPA	X-SAMPA	Description	Example	Viseme
h	h	voiceless glottal fricative	heili	k
j	j	palatal approximant	jökull	i
k ^h	k_h	aspirated voiceless velar plosive	ósköpunum	k
l	l	alveolar lateral approximant	gólf	t
ɭ	l_0	voiceless alveolar lateral approximant	fólk	t
m	m	bilabial nasal	september	p
ᵿ	m_0	voiceless bilabial nasal	kompá	p
n	n	alveolar nasal	númer	t
ᵿ	n_0	voiceless alveolar nasal	pöntun	t
ɲ	ɲ	palatal nasal	pælingar	ɲ
ŋ	ŋ	velar nasal	söngvarann	k
ᵿ	ŋ_0	voiceless velar nasal	frænka	k
p ^h	p_h	aspirated voiceless bilabial plosive	afplánun	p
r	r	alveolar trill	afskrifta	r

IPA	X-SAMPA	Description	Example	Viseme
ɾ	r_0	voiceless alveolar trill	andvörpum	r
s	s	voiceless alveolar fricative	baðhús	s
tʰ	t_h	aspirated voiceless alveolar plosive	tanki	t
θ	T	voiceless dental fricative	þeldökki	T
v	v	voiced labiodental fricative	silfur	f
w	w	labial-velar approximant		u
x	x	voiceless velar fricative	samfélags	k
Vowels				
œ	9	open-mid front rounded vowel	þröskuldinum	O
œ:	9:	long open-mid front rounded vowel	tvö	O
a	a	open front unrounded vowel	nefna	a
a:	a:	long open front unrounded vowel	fara	a
au	au	diphthong	átta	a

IPA	X-SAMPA	Description	Example	Viseme
au:	au:	diphthong	átján	a
ɛ	E	open-mid front unrounded vowel	kennari	E
ɛ:	E:	long open-mid front unrounded vowel	dreka	E
i	i	close front unrounded vowel	Gúlíver	i
i:	i:	long close front unrounded vowel	þrír	i
ɪ	ɪ	near-close near-front unrounded vowel	samspil	i
ɪ:	ɪ:	long near-close near-front unrounded vowel	stig	i
ɔ	O	open-mid back rounded vowel	regndropar	O
ɔ:	O:	long open-mid back rounded vowel	ullarbolur	O
ou	Ou	diphthong	tólf	O
ou:	Ou:	diphthong	fjórir	O
u	u	close back rounded vowel	stúlkan	u

IPA	X-SAMPA	Description	Example	Viseme
u:	u:	long close back rounded vowel	frú	u
ʏ	Y	near-close near-front rounded vowel	tíu	u
ʏ:	Y	long near-close near-front rounded vowel	gruninn	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Italian (it-IT)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Italian voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	b acca	p
d	d	voiced alveolar plosive	d ama	t

IPA	X-SAMPA	Description	Example	Viseme
ɖz	dz	voiced alveolar affricate	zero	s
ɖʒ	dʒ	voiced postalveolar affricate	giro	S
f	f	voiceless labiodental fricative	famiglia	f
g	g	voiced velar plosive	gatto	k
h	h	voiceless glottal fricative	horror	k
j	j	palatal approximant	dieci	i
k	k	voiceless velar plosive	campo	k
l	l	alveolar lateral approximant	lido	t
ʎ	ʎ	palatal lateral approximant	aglio	J
m	m	bilabial nasal	mille	p
n	n	alveolar nasal	nove	t
ɲ	ɲ	palatal nasal	lasagne	J
p	p	voiceless bilabial plosive	pizza	p
r	r	alveolar trill	risata	r

IPA	X-SAMPA	Description	Example	Viseme
s	s	voiceless alveolar fricative	sei	s
ʃ	S	voiceless postalveolar fricative	scienza	S
t	t	voiceless alveolar plosive	tavola	t
ʈs	ts	voiceless alveolar affricate	forza	s
ʈʃ	tS	voiceless postalveolar affricate	cielo	S
v	v	voiced labiodental fricative	venti	f
w	w	labial-velar approximant	quattro	u
z	z	voiced alveolar fricative	bisogno	s
ʒ	Z	voiced postalveolar fricative	bijou	S

Vowels

a	a	open front unrounded vowel	arco	a
e	e	close-mid front unrounded vowel	tre	e
ɛ	E	open-mid front unrounded vowel	ettaro	E

IPA	X-SAMPA	Description	Example	Viseme
i	i	close front unrounded vowel	impero	i
o	o	close-mid back rounded vowel	cento	o
ɔ	O	open-mid back rounded vowel	otto	O
u	u	close back rounded vowel	uno	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Japanese (ja-JP)

Amazon Polly supports the Pronunciation Kana and Yomigana alphabets for Japanese. To make Amazon Polly use phonetic pronunciation with these alphabets, use the phoneme alphabet="x-amazon-*phonetic standard used*" attribute.

- x-amazon-pron-kana – indicates that Pronunciation Kana is used. Pronunciation Kana are special Katakana characters used for phonetic transcription and can encode pitch accent.
- x-amazon-yomigana – indicates that Yomigana is used. Yomigana can be conventional Katakana, Hiragana, and Latin alphabets interpreted as hepburn romanization.

The following examples show how these are used:

Pronunciation Kana

```
<speaK>
```

```
###<phoneme alphabet="x-amazon-pron-kana" ph="###'#">##</phoneme>###
</speak>
```

Yomigana

```
<speak>
  ###<phoneme alphabet="x-amazon-yomigana" ph="####">##</phoneme>###
  ###<phoneme alphabet="x-amazon-yomigana" ph="####">##</phoneme>###
  ###<phoneme alphabet="x-amazon-yomigana" ph="Hirokazu">##</phoneme>###
</speak>
```

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Japanese voice supported by Amazon Polly.

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ɾ	4	alveolar flap	練習, renshuu	t
ʔ	ʔ	glottal stop	あっつ, atsu'	
b	b	voiced bilabial plosive	舞踊, buyou	p
β	B	voiced bilabial fricative	ヴィンテージ, vinteeji	B
c	c	voiceless palatal plosive	ききょう, kikyō	k
ç	C	voiceless palatal fricative	人, hito	k
d	d	voiced alveolar plosive	濁点, dakuten	t
ɸɖ͡	dz\	voiced alveolo-palatal affricate	純, jun	J

IPA	X-SAMPA	Description	Example	Viseme
g	g	voiced velar plosive	ご飯, gohan	k
h	h	voiceless glottal fricative	本, hon	k
j	j	palatal approximant	屋根, yane	i
ɟ	J\	voiced palatal plosive	行儀, gyougi	J
k	k	voiceless velar plosive	漢字, kanji	k
ɭ	l\	alveolar lateral flap	釣り, tsuri	r
ɭj	l\j	alveolar lateral flap, palatal approximant	流行, ryuukou	r
m	m	bilabial nasal	飯, meshi	p
n	n	alveolar nasal	猫, neko	t
ɲ	J	palatal nasal	日本, nippon	J
ɴ	N\	uvular nasal	缶, kan	k
p	p	voiceless bilabial plosive	パン, pan	p
ɸ	p\	voiceless bilabial fricative	福, huku	f
s	s	voiceless alveolar fricative	層, sou	s

IPA	X-SAMPA	Description	Example	Viseme
ɸ	s\	voiceless alveolo-palatal fricative	書簡, shokan	J
t	t	voiceless alveolar plosive	手紙, tegami	t
ʈs	ts	voiceless alveolar affricate	釣り, tsuri	s
ʈɕ	ts\	voiceless alveolo-palatal affricate	吉, kichi	J
w	w	labial-velar approximant	電話, denwa	u
z	z	voiced alveolar fricative	座敷, zashiki	s
Vowels				
ä:	a:_"	long open central unrounded vowel	羽蟻, haari	a
ä	a_"	open central unrounded vowel	仮名, kana	a
e:	e:_o	long mid front unrounded vowel	学生, gakusei	@
e	e_o	mid front unrounded vowel	歴, reki	@
i	i	close front unrounded vowel	気, ki	i
i:	i:	long close front unrounded vowel	詩歌, shiika	i

IPA	X-SAMPA	Description	Example	Viseme
ʊ	M	close back unrounded vowel	運, un	i
ʊ:	M:	long close back unrounded vowel	宗教, shuukyou	i
o:	o:_o	long mid back rounded vowel	購読, koodoku	o
o	o_o	mid back rounded vowel	読者, dokusha	o

Korean (ko-KR)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA)symbols, and the corresponding visemes for the Korean voice supported by Amazon Polly.

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
k	k	voiceless velar plosive	강, [g]ang	k
k#	k_t	strong voiceless velar plosive	깨, [kk]e	k
n	n	alveolar nasal	남, [n]am	t
t	t	voiceless alveolar plosive	도, [d]o	t
t#	t_t	strong voiceless alveolar plosive	때, [tt]e	t

IPA	X-SAMPA	Description	Example	Viseme
r	4	alveolar flap	사랑, sa[r]ang	t
l	l	alveolar lateral approximant	돌, do[l]	t
m	m	bilabial nasal	무, [m]u	p
p	p	voiceless bilabial plosive	봄, [b]om	p
p#	p_t	strong voiceless bilabial plosive	빨, [pp]eol	p
s	s	voiceless alveolar fricative	새, [s]e	s
s#	s_t	strong voiceless alveolar fricative	씨, [ss]i	s
ŋ	N	velar nasal	방, ba[ng]	k
t͡ɕ	ts\	voiceless alveolo-palatal affricate	조, [j]o	J
t͡ɕʰ	ts_t	strong voiceless alveolo-palatal affricate	찌, [jj]i	J
t͡ɕʰ	ts_h	aspirated voiceless alveolo-palatal affricate	차, [ch]a	J
kʰ	k_h	aspirated voiceless velar plosive	코, [k]o	k
tʰ	t_h	aspirated voiceless alveolar plosive	통, [t]ong	t

IPA	X-SAMPA	Description	Example	Viseme
p ^h	p_h	aspirated voiceless bilabial plosive	패, [p]e	p
h	h	voiceless glottal fricative	힘, [h]im	k
j	j	palatal approximant	양, [y]ang	i
w	w	labial-velar approximant	왕, [w]ang	u
ɰ	M\	velar approximant>	의, [w]i	i

Vowels

a	a	open front unrounded vowel	밥, b[a]b	a
ʌ	V	open-mid back unrounded vowel	정, j[eo]ng	E
ɛ	E	open-mid front unrounded vowel	배, b[e]	E
o	o	close-mid back rounded vowel	노, n[o]	o
u	u	close back rounded vowel	둘, d[u]l	u
ɯ	M	close back unrounded vowel	은, [eu]n	i
i	i	close front unrounded vowel	김, k[i]m	i

Norwegian (nb-NO)

The following chart lists the full set of International Phonetic Alphabet (IPA) phonemes and the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols as well as the corresponding visemes as supported by Amazon Polly for Norwegian language voices.

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
r	4	alveolar flap	prøv	t
b	b	voiced bilabial plosive	lab b	p
ç	C	voiceless palatal fricative	kino	k
d	d	voiced alveolar plosive	ladd	t
ɖ	d`	voiced retroflex plosive	ver di	t
f	f	voiceless labiodental fricative	fot	f
g	g	voiced velar plosive	tag g	k
h	h	voiceless glottal fricative	ha	k
j	j	palatal approximant	gi	i
k	k	voiceless velar plosive	tak k	k

IPA	X-SAMPA	Description	Example	Viseme
l	l	alveolar lateral approximant	fall, ball	t
ɭ	l`	retroflex lateral approximant	ærlig	t
m	m	bilabial nasal	lam	p
n	n	alveolar nasal	vann	t
ɳ	n`	retroflex nasal	garn	t
ŋ	N	velar nasal	sang	k
p	p	voiceless bilabial plosive	hopp	p
s	s	voiceless alveolar fricative	lass	s
ʂ	s`	voiceless retroflex fricative	års	S
ʃ	S	voiceless postalveolar fricative	skyt	S
t	t	voiceless alveolar plosive	lat	t
ɖ	t`	voiceless retroflex plosive	hardt	t
ʋ	v\	labiodental approximant	vin	f
w	w	labial-velar approximant	will	x

IPA	X-SAMPA	Description	Example	Viseme
Vowels				
ø:	2:	long close-mid front rounded vowel	søt	o
œ	9	open-mid front rounded vowel	søtt	O
ə	@	mid central vowel	ape	@
æ:	{:	long near-open front unrounded vowel	vær	a
ʊ	}	close central rounded vowel	lund	u
ʊ:	}::	long close central rounded vowel	lun	u
æ	{	near-open front unrounded vowel	vært	a
ɑ	A	open back unrounded vowel	hatt	a
ɑ:	A:	long open back unrounded vowel	hat	a
e:	e:	long close-mid front unrounded vowel	sen	e
ɛ	E	open-mid front unrounded vowel	send	E

IPA	X-SAMPA	Description	Example	Viseme
i:	i:	long close front unrounded vowel	vin	i
ɪ	ɪ	near-close near-front unrounded vowel	vind	i
o:	o:	long close-mid back rounded vowel	våt	o
ɔ	ɔ	open-mid back rounded vowel	vått	ɔ
u:	u:	long close back rounded vowel	bok	u
ʊ	ʊ	near-close near-back rounded vowel	bukk	u
y:	y:	long close front rounded vowel	lyn	u
ʏ	ʏ	near-close near-front rounded vowel	lynne	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Polish (pl-PL)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Polish voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bobas, belka	p
d	d	voiced alveolar plosive	dar, do	t
ɖz	dz	voiced alveolar affricate	dzwon, widzowie	s
ɖʑ	dz\	voiced alveolo-palatal affricate	dźwięk	j
ɖʐ	dz `	voiced retroflex affricate	dżem, dżungla	S
f	f	voiceless labiodental fricative	furtka, film	f
g	g	voiced velar plosive	gazeta, waga	k
h	h	voiceless glottal fricative	chleb, handel	k
j	j	palatal approximant	jak, maja	i

IPA	X-SAMPA	Description	Example	Viseme
k	k	voiceless velar plosive	kura, marek	k
l	l	alveolar lateral approximant	lipa, alicja	t
m	m	bilabial nasal	matka, molo	p
n	n	alveolar nasal	norka	t
ɲ	J	palatal nasal	koń, toruń	J
p	p	voiceless bilabial plosive	pora, stop	p
r	r	alveolar trill	rok, park	r
s	s	voiceless alveolar fricative	sum, pas	s
ɕ	s\	voiceless alveolo-palatal fricative	śruba, śnieg	J
ʂ	s`	voiceless retroflex fricative	szum, masz	S
t	t	voiceless alveolar plosive	tok, stół	t
ʈs	ts	voiceless alveolar affricate	car, co	s
ʈɕ	ts\	voiceless alveolo-palatal affricate	ćma, mieć	J
ʈʂ	ts`	voiceless retroflex affricate	czas, raczej	S

IPA	X-SAMPA	Description	Example	Viseme
v	v	voiced labiodental fricative	worek, mewa	f
w	w	labial-velar approximant	łaska, mało	u
z	z	voiced alveolar fricative	zero	s
ʐ	z\	voiced alveolo-palatal fricative	żrebie, bieliźnie	J
ʐ̤	z`	voiced retroflex fricative	żar, żona	S
Vowels				
a	a	open front unrounded vowel	ja	a
ɛ	E	open-mid front unrounded vowel	echo	E
ɛ̃	E~	nasal open-mid front unrounded vowel	węże	E
i	i	close front unrounded vowel	ile	i
ɔ	O	open-mid back rounded vowel	oczy	O
ɔ̃	O~	nasal open-mid back rounded vowel	wąż	O

IPA	X-SAMPA	Description	Example	Viseme
u	u	close back rounded vowel	uczta	u
ɨ	ɨ	close central unrounded vowel	byk	ɨ

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Portuguese (pt-PT)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Portuguese voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ɾ	ɾ	alveolar flap	pira	t
b	b	voiced bilabial plosive	d ato	p
d	d	voiced alveolar plosive	d ato	t
f	f	voiceless labiodental fricative	f acto	f

IPA	X-SAMPA	Description	Example	Viseme
g	g	voiced velar plosive	g ato	k
j	j	palatal approximant	para j uay	i
k	k	voiceless velar plosive	k ato	k
l	l	alveolar lateral approximant	g alo	t
ʎ	L	palatal lateral approximant	g alho	J
m	m	bilabial nasal	m ato	p
n	n	alveolar nasal	n ato	t
ɲ	J	palatal nasal	p inha	J
p	p	voiceless bilabial plosive	p ato	p
ʀ	R\	uvular trill	b arroso	k
s	s	voiceless alveolar fricative	s aca	s
ʃ	S	voiceless postalveolar fricative	ch ato	S
t	t	voiceless alveolar plosive	t ato	t
v	v	voiced labiodental fricative	v aca	f

IPA	X-SAMPA	Description	Example	Viseme
w	w	labial-velar approximant	mau	u
z	z	voiced alveolar fricative	zaca	s
ʒ	Z	voiced postalveolar fricative	jacto	S

Vowels

a	a	open front unrounded vowel	parto	a
a~	a~	nasal open front unrounded vowel	pega	a
e	e	close-mid front unrounded vowel	pega	e
e~	e~	nasal close-mid front unrounded vowel	movem	e
ɛ	E	open-mid front unrounded vowel	café	E
i	i	close front unrounded vowel	lingueta	i
ĩ	i~	nasal close front unrounded vowel	cinto	i
o	o	close-mid back rounded vowel	poder	o

IPA	X-SAMPA	Description	Example	Viseme
õ	õ	nasal close-mid back rounded vowel	compra	o
ɔ	O	open-mid back rounded vowel	cotó	O
u	u	close back rounded vowel	fui	u
ũ	ũ	nasal close back rounded vowel	sunto	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Portuguese (Brazilian) (pt-BR)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Brazilian Portuguese voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
r	4	alveolar flap	pira	t
b	b	voiced bilabial plosive	b ato	p

IPA	X-SAMPA	Description	Example	Viseme
d	d	voiced alveolar plosive	dato	t
ɖʒ	dʒ	voiced postalveolar affricate	idade	S
f	f	voiceless labiodental fricative	facto	f
g	g	voiced velar plosive	gato	k
j	j	palatal approximant	paraguay	i
k	k	voiceless velar plosive	cacto	k
l	l	alveolar lateral approximant	galo	t
ʎ	ʎ	palatal lateral approximant	galho	J
m	m	bilabial nasal	mato	p
n	n	alveolar nasal	nato	t
ɲ	ɲ	palatal nasal	pinha	J
p	p	voiceless bilabial plosive	pato	p
s	s	voiceless alveolar fricative	saca	s
ʃ	ʃ	voiceless postalveolar fricative	chato	S

IPA	X-SAMPA	Description	Example	Viseme
t	t	voiceless alveolar plosive	tacto	t
ʈʃ	tʂ	voiceless postalveolar affricate	noite	ʂ
v	v	voiced labiodental fricative	vaca	f
w	w	labial-velar approximant	mau	u
χ	X	voiceless uvular fricative	carro	k
z	z	voiced alveolar fricative	zaca	s
ʒ	ʒ	voiced postalveolar fricative	jacto	ʂ

Vowels

a	a	open front unrounded vowel	parto	a
ã	ã	nasal open front unrounded vowel	pensamos	a
e	e	close-mid front unrounded vowel	pega	e
ẽ	ẽ	nasal close-mid front unrounded vowel	movem	e
ɛ	ɛ	open-mid front unrounded vowel	café	ɛ

IPA	X-SAMPA	Description	Example	Viseme
i	i	close front unrounded vowel	lingueta	i
ĩ	i~	nasal close front unrounded vowel	cinto	i
o	o	close-mid back rounded vowel	poder	o
o~	o~	nasal close-mid back rounded vowel	compra	o
ɔ	O	open-mid back rounded vowel	cotó	O
u	u	close back rounded vowel	fui	u
u~	u~	nasal close back rounded vowel	sunto	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Romanian (ro-RO)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Romanian voice supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bubă	p
d	d	voiced alveolar plosive	după	t
dʒ	dʒ	voiced postalveolar affricate	george	s
f	f	voiceless labiodental fricative	afacere	f
g	g	voiced velar plosive	agriș	k
h	h	voiceless glottal fricative	harpă	k
j	j	palatal approximant	baie	i
k	k	voiceless velar plosive	coș	k
l	l	alveolar lateral approximant	lampa	t
m	m	bilabial nasal	mama	p
n	n	alveolar nasal	nor	t
p	p	voiceless bilabial plosive	pilă	p
r	r	alveolar trill	rampă	r

IPA	X-SAMPA	Description	Example	Viseme
s	s	voiceless alveolar fricative	soare	s
ʃ	S	voiceless postalveolar fricative	mașină	S
t	t	voiceless alveolar plosive	tata	t
ʈs	ts	voiceless alveolar affricate	țară	s
ʈʃ	tS	voiceless postalveolar affricate	ceai	S
v	v	voiced labiodental fricative	viață	f
w	w	labial-velar approximant	beau	u
z	z	voiced alveolar fricative	mozol	s
ʒ	Z	voiced postalveolar fricative	joacă	S
Vowels				
ə	@	mid central vowel	babă	@
a	a	open front unrounded vowel	casa	a
e	e	close-mid front unrounded vowel	elan	e

IPA	X-SAMPA	Description	Example	Viseme
ɐ̯	e_^	non-syllabic close-mid front unrounded vowel	beau	e
i	i	close front unrounded vowel	mie	i
o	o	close-mid back rounded vo	oră	o
oa	o_^a	diphthong	oare	o
u	u	close back rounded vowel	unde	u
ɨ	1	close central unrounded vowel	România	i
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Russian (ru-RU)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Russian voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				

IPA	X-SAMPA	Description	Example	Viseme
b	b	voiced bilabial plosive	борт	p
bʲ	b'	palatalized voiced bilabial plosive	бюро	p
d	d	voiced alveolar plosive	дом	t
dʲ	d'	palatalized voiced alveolar plosive	дядя	t
f	f	voiceless labiodental fricative	флаг	f
fʲ	f'	palatalized voiceless labiodental fricative	февраль	f
g	g	voiced velar plosive	нога	k
gʲ	g'	palatalized voiced velar plosive	герой	k
j	j	palatal approximant	дизайн, ящик	i
k	k	voiceless velar plosive	кот	k
kʲ	k'	palatalized voiceless velar plosive	кино	k
l	l	alveolar lateral approximant	лампа	t

IPA	X-SAMPA	Description	Example	Viseme
lʲ	l'	palatalized alveolar lateral approximant	лес	t
m	m	bilabial nasal	мама	p
mʲ	m'	palatalized bilabial nasal	мяч	p
n	n	alveolar nasal	нос	t
nʲ	n'	palatalized alveolar nasal	няня	t
p	p	voiceless bilabial plosive	папа	p
pʲ	p'	palatalized voiceless bilabial plosive	перо	p
r	r	alveolar trill	роза	r
rʲ	r'	palatalized alveolar trill	рюмка	r
s	s	voiceless alveolar fricative	сыр	s
sʲ	s'	palatalized voiceless alveolar fricative	сердце, русь	s
ɕ:	s\:	long voiceless alveolo-palatal fricative	щека	J

IPA	X-SAMPA	Description	Example	Viseme
ʂ	s`	voiceless retroflex fricative	шум	S
t	t	voiceless alveolar plosive	точка	t
tʲ	t'	palatalized voiceless alveolar plosive	тётя	t
ʈs	ts	voiceless alveolar affricate	царь	s
ʈʂ	ts\	voiceless alveolo-palatal affricate	час	J
v	v	voiced labiodental fricative	вор	f
vʲ	v'	palatalized voiced labiodental fricative	верфь	f
x	x	voiceless velar fricative	хор	k
xʲ	x'	palatalized voiceless velar fricative	химия	k
z	z	voiced alveolar fricative	зуб	s
zʲ	z'	palatalized voiced alveolar fricative	зима	s

IPA	X-SAMPA	Description	Example	Viseme
ʐ:	z\:	long voiced alveolo-palatal fricative	уезжать	J
ʐ	z`	voiced retroflex fricative	жена	S

Vowels

ə	@	mid central vowel	канарейка	@
a	a	open front unrounded vowel	два, яблоко	a
e	e	close-mid front unrounded vowel	печь	e
ɛ	E	open-mid front unrounded vowel	это	E
i	i	close front unrounded vowel	один, четыре	i
o	o	close-mid back rounded vowel	кот	o
u	u	close back rounded vowel	муж, вьюга	u
ɨ	1	close central unrounded vowel	мышь	i

Spanish (es-ES)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Spanish voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
ɾ	4	alveolar flap	pero, bravo, amor, eterno	t
b	b	voiced bilabial plosive	bestia	p
β	B	voiced bilabial fricative	bebé	B
d	d	voiced alveolar plosive	cuando	t
ð	D	voiced dental fricative	arder	T
f	f	voiceless labiodent al fricative	fase, café	f
g	g	voiced velar plosive	gato, lengua, guerra	k
ɣ	G	voiced velar fricative	trigo, Argos	k
j	j	palatal approxima nt	hacia, tierra, radio, viuda	i
ɟ	j\	voiced palatal fricative	enhielar, sayo, inyectado, desyerba	J
k	k	voiceless velar plosive	caña, laca, quisimos	k

IPA	X-SAMPA	Description	Example	Viseme
l	l	alveolar lateral approximant	lino, calor, principal	t
ʎ	L	palatal lateral approximant	llave, pollo	J
m	m	bilabial nasal	madre, comer, anfibio	p
n	n	alveolar nasal	nido, anillo, sin	t
ɲ	J	palatal nasal	cabaña, ñoquis	J
ŋ	N	velar nasal	cinco, venga	k
p	p	voiceless bilabial plosive	pozo, topo	p
r	r	alveolar trill	perro, enrachado	r
s	s	voiceless alveolar fricative	saco, casa, puertas	s
t	t	voiceless alveolar plosive	tamiz, átomo	t
ʈʂ	tS	voiceless postalveolar affricate	chubasco	S
θ	T	voiceless dental fricative	cereza, zorro, lacero, paz	T
w	w	labial-velar approximant	fuego, fuimos, cuota, cuadro	u
x	x	voiceless velar fricative	jamón, general, suje, reloj	k

IPA	X-SAMPA	Description	Example	Viseme
z	z	voiced alveolar fricative	rasgo, mismo	s
Vowels				
a	a	open front unrounded vowel	tanque	a
e	e	close-mid front unrounded vowel	peso	e
i	i	close front unrounded vowel	cinco	i
o	o	close-mid back rounded vowel	bosque	o
u	u	close-mid front unrounded vowel	publicar	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Spanish (Mexican) (es-MX)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Mexican Spanish voice that is supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
r	4	alveolar flap	pero, bravo, amor, eterno	t
b	b	voiced bilabial plosive	bestia	p
β	B	voiced bilabial fricative	bebé	B
d	d	voiced alveolar plosive	cuando	t
ð	D	voiced dental fricative	arder	T
f	f	voiceless labiodental fricative	fase, café	f
g	g	voiced velar plosive	gato, lengua, guerra	k
ɣ	G	voiced velar fricative	trigo, Argos	k
j	j	palatal approximant	hacia, tierra, radio, viuda	i
ɟ	j\	voiced palatal fricative	enhielar, sayo, inyectado, desyerba	J
k	k	voiceless velar plosive	caña, laca, quisimos	k

IPA	X-SAMPA	Description	Example	Viseme
l	l	lateral alveolar approximant	lino, calor, principal	t
m	m	bilabial nasal	madre, comer, anfibio	p
n	n	alveolar nasal	nido, anillo, sin	t
ɲ	J	palatal nasal	cabaña, ñoquis	J
ŋ	N	velar nasal	angosto, increíble	k
p	p	voiceless bilabial plosive	pozo, topo	p
r	r	alveolar trill	perro, enrachado	r
s	s	voiceless alveolar fricative	saco, casa, puertas	s
ʃ	S	voiceless postalveolar fricative	show, flash	S
t	t	voiceless alveolar plosive	tamiz, átomo	t
tʃ	tS	voiceless postalveolar affricate	chubasco	S
w	w	labial-velar approximant	fuego, fuimos, cuota, cuadro	u
x	x	voiceless velar fricative	jamón, general, peaje, reloj	k
z	z	voiced alveolar fricative	rasgo, mismo	s

IPA	X-SAMPA	Description	Example	Viseme
Vowels				
a	a	central open unrounded vowel	tanque	a
e	e	close-mid front unrounded vowel	peso	e
i	i	close front unrounded vowel	cinco	i
o	o	close-mid back rounded vowel	bosque	o
u	u	close back rounded vowel	publicar	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Spanish (US) (es-US)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the US Spanish voices that are supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				

IPA	X-SAMPA	Description	Example	Viseme
r	4	alveolar flap	pero, bravo, amor, eterno	t
b	b	voiced bilabial plosive	bestia	p
β	B	voiced bilabial fricative	bebé	B
d	d	voiced alveolar plosive	cuando	t
ð	D	voiced dental fricative	arder	T
f	f	voiceless labiodental fricative	fase, café	f
g	g	voiced velar plosive	gato, lengua, guerra	k
ɣ	G	voiced velar fricative	trigo, Argos	k
j	j	palatal approximant	hacia, tierra, radio, viuda	i
ɟ	j\	voiced palatal fricative	enhielar, sayo, inyectado, desyerba	J
k	k	voiceless velar plosive	caña, laca, quisimos	k
l	l	lateral alveolar approximant	lino, calor, principal	t

IPA	X-SAMPA	Description	Example	Viseme
m	m	bilabial nasal	madre, comer, anfibio	p
n	n	alveolar nasal	nido, anillo, sin	t
ɲ	J	palatal nasal	cabaña, ñoquis	J
ŋ	N	velar nasal	angosto, increíble	k
p	p	voiceless bilabial plosive	pozo, topo	p
r	r	alveolar trill	perro, enrachado	r
s	s	voiceless alveolar fricative	saco, casa, puertas	s
ʃ	S	voiceless postalveolar fricative	show, flash	S
t	t	voiceless alveolar plosive	tamiz, átomo	t
tʃ	tS	voiceless postalveolar affricate	chubasco	S
w	w	labial-velar approximant	fuego, fuimos, cuota, cuadro	u
x	x	voiceless velar fricative	jamón, general, peaje, reloj	k
z	z	voiced alveolar fricative	rasgo, mismo	s

Vowels

IPA	X-SAMPA	Description	Example	Viseme
a	a	central open unrounded vowel	tanque	a
e	e	close-mid front unrounded vowel	peso	e
i	i	close front unrounded vowel	cinco	i
o	o	close-mid back rounded vowel	bosque	o
u	u	close back rounded vowel	publicar	u

Additional Symbols

'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Swedish (sv-SE)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Swedish voice supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	voiced bilabial plosive	bil	p

IPA	X-SAMPA	Description	Example	Viseme
d	d	voiced alveolar plosive	dal	t
ɖ	d`	voiced retroflex plosive	bord	t
f	f	voiceless labiodental fricative	fil	f
g	g	voiced velar plosive	gå s	k
h	h	voiceless glottal fricative	hal	k
j	j	palatal approximant	jag	i
k	k	voiceless velar plosive	kal	k
l	l	alveolar lateral approximant	lös	t
ɭ	l`	retroflex lateral approximant	härlig	t
m	m	bilabial nasal	mil	p
n	n	alveolar nasal	nålar	t
ɳ	n`	retroflex nasal	barn	t
ŋ	N	velar nasal	ring	k
p	p	voiceless bilabial plosive	pil	p

IPA	X-SAMPA	Description	Example	Viseme
r	r	alveolar trill	ris	r
s	s	voiceless alveolar fricative	sil	s
ʃ	s\	voiceless alveolo-palatal fricative	tjock	J
ʂ	s`	voiceless retroflex fricative	fors, schlager	S
t	t	voiceless alveolar plosive	tal	t
ʈ	t`	voiceless retroflex plosive	hjort	t
v	v	voiced labiodental fricative	vår	f
w	w	labial-velar approximant	aula, airways	u
ɣ	x\	voiceless palatal-velar fricative	sjuk	k
Vowels				
ø	2	close-mid front rounded vowel	föll, förr	o
ø	2:	long close-mid front rounded vowel	föl, nöt, för	o
ɵ	8	close-mid central rounded vowel	buss, full	o

IPA	X-SAMPA	Description	Example	Viseme
ə	@	mid central vowel	pojken	@
ʊ:	}:	long close central rounded vowel	hus, ful	u
a	a	open front unrounded vowel	hall, matt	a
æ	{	near-open front unrounded vowel	herr	a
ɑ:	A:	long open back unrounded vowel	hal, mat	a
e:	e:	long close-mid front unrounded vowel	vet, hel	e
ɛ	E	open-mid front unrounded vowel	vett, rätt, hetta, håll	E
ɛ:	E:	long open-mid front unrounded vowel	säl, häl, här	E:
i:	i:	long close front unrounded vowel	vit, sil	i:
ɪ	ɪ	near-close near-front unrounded vowel	vitt, sill	i
o:	o:	long close-mid back rounded vowel	håll, mål	o

IPA	X-SAMPA	Description	Example	Viseme
ɔ	O	open-mid back rounded vowel	håll, moll	O
u:	u:	long close back rounded vowel	sol, bot	u
ʊ	U	near-close near-back rounded vowel	bott	u
y	y	close front rounded vowel	bytt	u
y:	y:	long close front rounded vowel	syl, syl	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Turkish (tr-TR)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Turkish voice supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
r	4	alveolar flap	durum	t

IPA	X-SAMPA	Description	Example	Viseme
ɾ	4_0_r	voiceless fricated alveolar flap	bir	t
ɹ	4_r	fricated alveolar flap	raf	t
b	b	voiced bilabial plosive	raf	p
c	c	voiceless palatal plosive	kedı	k
d	d	voiced alveolar plosive	dede	t
ɟ͡ʒ	dʒ	voiced postalveolar affricate	cam	s
f	f	voiceless labiodental fricative	fare	f
g	g	voiced velar plosive	galibi	k
h	h	voiceless glottal fricative	hasta	k
j	j	palatal approximant	yat	i
ɟ	ʝ\	voiced palatal plosive	genç	j
k	k	voiceless velar plosive	akıl	k
l	l	alveolar lateral approximant	lale	t

IPA	X-SAMPA	Description	Example	Viseme
ɫ	5	velarized alveolar lateral approximant	labirent	t
m	m	bilabial nasal	maaş	p
n	n	alveolar nasal	anı	t
p	p	voiceless bilabial plosive	ip	p
s	s	voiceless alveolar fricative	ses	s
ʃ	S	voiceless postalveolar fricative	aşı	S
t	t	voiceless alveolar plosive	ütü	t
tʃ	tS	voiceless postalveolar affricate	çaba	S
v	v	voiced labiodental fricative	ekvator, kahveci, akvaryum, isveçli, teşviki, cetvel	f
z	z	voiced alveolar fricative	ver	s
ʒ	Z	voiced postalveolar fricative	azık	S
Vowels				
ø	2	close-mid front rounded vowel	göl	0

IPA	X-SAMPA	Description	Example	Viseme
œ	9	open-mid front rounded vowel	banliyö	O
a	a	open front unrounded vowel	kal	a
a:	a:	long open front unrounded vowel	davacı	a
æ	{	near-open front unrounded vowel	özlem, güvenlik, gürel, somersault	a
e	e	close-mid front unrounded vowel	keçi	e
ɛ	E	open-mid front unrounded vowel	dede	E
i	i	close front unrounded vowel	bir	i
i:	i:	long close front unrounded vowel	izah	i
ɪ	ɪ	near-close near-front unrounded vowel	keçi	i
ʊ	ʊ	close back unrounded vowel	kıl	i
o	o	close-mid back rounded vowel	kol	o
o:	o:	long close-mid back rounded vowel	dolar	o

IPA	X-SAMPA	Description	Example	Viseme
u	u	close back rounded vowel	dur <u>u</u> m	u
u:	u:	long close back rounded vowel	ru <u>u</u> m	u
ʊ	U	near-close near-back rounded vowel	do <u>u</u>	u
y	y	close front rounded vowel	g <u>y</u> venlik	u
ɣ	Y	near-close near-front rounded vowel	a <u>ɣ</u> i	u
Additional Symbols				
'	"	primary stress	Alab <u>a</u> ma	
,	%	secondary stress	<u>A</u> labama	
.	.	syllable boundary	A.la.ba.ma	

Welsh (cy-GB)

The following table lists the International Phonetic Alphabet (IPA) phonemes, the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols, and the corresponding visemes for the Welsh voice supported by Amazon Polly.

Phoneme/Viseme Table

IPA	X-SAMPA	Description	Example	Viseme
Consonants				

IPA	X-SAMPA	Description	Example	Viseme
b	b	voiced bilabial plosive	baban	p
d	d	voiced alveolar plosive	deg	t
ɖʒ	dʒ	voiced postalveolar affricate	garej	S
ð	D	voiced dental fricative	deuddeg	T
f	f	voiceless labiodental fricative	ffacs	f
g	g	voiced velar plosive	gadael	k
h	h	voiceless glottal fricative	haearn	k
j	j	palatal approximant	astudio	i
k	k	voiceless velar plosive	cant	k
l	l	alveolar lateral approximant	lan	t
ɬ	K	voiceless alveolar lateral fricative	llan	t
m	m	bilabial nasal	mae	p
ɱ	m_0	voiceless bilabial nasal	ymhen	p

IPA	X-SAMPA	Description	Example	Viseme
n	n	alveolar nasal	naw	t
ɳ	n_0	voiceless alveolar nasal	anhawster	t
ŋ	N	velar nasal	argyfwng	k
ŋ̥	N_0	voiceless velar nasal	anghenion	k
p	p	voiceless bilabial plosive	pump	p
r	r	alveolar trill	rhoi	r
ɾ	r_0	voiceless alveolar trill	garw	r
s	s	voiceless alveolar fricative	saith	s
ʃ	S	voiceless postalveolar fricative	siawns	S
t	t	voiceless alveolar plosive	tegan	t
tʃ	tS	voiceless postalveolar affricate	cytsain	S
θ	T	voiceless dental fricative	aberth	T
v	v	voiced labiodental fricative	prawf	f
w	w	labial-velar approximant	rhagweld	u

IPA	X-SAMPA	Description	Example	Viseme
χ	X	voiceless uvular fricative	chwech	k
z	z	voiced alveolar fricative	aids	s
ʒ	Z	voiced postalveolar fricative	rouge	S
Vowels				
ə	@	mid central vowel	ychwanega	@
a	a	open front unrounded vowel	acen	a
ai	ai	diphthong	dau	a
au	au	diphthong	awdur	a
ɑ:	A:	long open back unrounded vowel	mab	a
ɑ:ɨ	A:1	diphthong	aelod	a
e:	e:	long close-mid front unrounded vowel	peth	e
ɛ	E	open-mid front unrounded vowel	pedwar	E
ɛi	Ei	diphthong	beic	E
i:	i:	long close front unrounded vowel	tri	i

IPA	X-SAMPA	Description	Example	Viseme
ɪ	ɪ	near-close near-front unrounded vowel	miliwn	i
ɨu	ɨu	diphthong	unigryw	i
o:	o:	long close-mid back rounded vowel	oddi	o
ɔ	O	open-mid back rounded vowel	oddieithr	O
ɔi	Oi	diphthong	troi	O
ɔu	Ou	diphthong	rownd	O
u:	u:	long close back rounded vowel	cwch	u
ʊ	U	near-close near-back rounded vowel	acwstig	u
ʊi	Ui	diphthong	wyth	u
Additional Symbols				
'	"	primary stress	Alab a ma	
,	%	secondary stress	A labama	
.	.	syllable boundary	A.la.ba.ma	

Amazon Polly voice engines

Amazon Polly has four voice engines that convert input text into life-like speech. These include: **Generative**, **Long-form**, **Neural**, and **Standard**. To use an Amazon Polly voice, select an engine and a speech synthesis API operation. Then provide input text for the engine to synthesize, and select an audio output format. Given these inputs, Amazon Polly synthesizes the provided text into a high-quality speech audio stream.

The following sections include details about the voice engines offered by Amazon Polly.

Topics

- [Generative voices](#)
- [Long-form voices](#)
- [Neural voices](#)
- [Standard voices](#)
- [Choosing a voice engine](#)

Generative voices

Amazon Polly's **generative** text-to-speech (TTS) engine offers the most human-like, emotionally engaged, and adaptive conversational voices available for the use via the Amazon Polly console.

The **Generative engine** is the largest Amazon Polly TTS model to-date. It deploys a billion-parameter transformer that converts raw text into speech codes, followed by a convolution-based decoder that converts these speech codes into waveforms in an incremental, streamable manner. This method shows the widely-reported emergent abilities of Large Language Models (LLMs) when trained on increasing volumes of publicly available and proprietary data comprising a variety of voices, languages, and styles.

The Generative engine creates synthetic speech which is emotionally engaged, assertive, and highly colloquial in a way that is remarkably similar to a human voice. You can use these voices as a knowledgeable customer assistant, a virtual trainer, or an advertiser with a near-human synthetic speech.

Note

The state-of-the-art technology underlying these voices falls within the paradigm of generative AI for language and voice modelling. A side effect of the technology is that any updates to the training data and the model could result in slight variations to the way the voices sound, even in case when their overall quality improves with model updates. This could have an impact on use cases with different content parts synthesized over a long time period – for example, a season of podcasts.

Available generative voices

Amazon Polly currently offers 43 voices in a generative variant.

	Language	Language code	Name/ID	Gender
1	English (Australian)	en-AU	Olivia	Female
2	English (British)	en-GB	Amy	Female
			Brian	Male
3	English (Indian)	en-IN	Kajal	Female
4	English (Ireland)	en-IE	Niamh	Female
5	English (New Zealand)	en-NZ	Aria	Female
6	English (Singaporean)	en-SG	Jasmine	Female
7	English (South African)	en-ZA	Ayanda	Female
8	English (US)	en-US	Danielle	Female
			Joanna	Female

	Language	Language code	Name/ID	Gender
			Matthew	Male
			Ruth	Female
			Salli	Female
			Stephen	Male
			Tiffany	Female
9	Dutch (Belgium)	nl-BE	Lisa	Female
10	Dutch (Netherlands)	nl-NL	Laura	Female
11	French (Belgian)	fr-BE	Isabelle	Female
12	French (Canadian)	fr-CA	Gabrielle	Female
			Liam	Male
13	French (France)	fr-FR	Ambre	Female
			Céline	Female
			Florian	Male
			Léa	Female
			Rémi	Male
14	German (Austria)	de-AT	Hannah	Female
15	German (Germany)	de-DE	Daniel	Male
			Lennart	Male
			Vicki	Female
16	German (Swiss)	de-CH	Sabrina	Female

	Language	Language code	Name/ID	Gender
17	Italian (Italy)	it-IT	Beatrice	Female
			Bianca	Female
			Lorenzo	Male
18	Korean (Korea)	ko-KR	Seoyeon	Female
19	Polish (Poland)	pl-PL	Ewa	Female
			Ola	Female
20	Portuguese (Brazilian)	pt-BR	Camila	Female
21	Spanish (Mexican)	es-MX	Andrés	Male
			Mía	Female
22	Spanish (Spain)	es-ES	Lucia	Female
			Sergio	Male
23	Spanish (US)	es-US	Lupe	Female
			Pedro	Male

Note

Generative voices cost is specified on the [Amazon Polly pricing information page](#).

Feature and region compatibility

Amazon Polly generative voices are available in the following regions:

- US East (N. Virginia): us-east-1
- Europe (Frankfurt): eu-central-1

- US West (Oregon): us-west-2
- Asia Pacific (Tokyo): ap-northeast-1
- Asia Pacific (Seoul): ap-northeast-2
- Asia Pacific (Singapore): ap-southeast-1
- Asia Pacific (Sydney): ap-southeast-2
- Europe (London): eu-west-2
- Canada (Central): ca-central-1
- Europe (Zurich): eu-central-2
- Other Regions are not available

The following features are supported for generative voices:

- Bidirectional Streaming API is now offered in Generative engine and allows for streaming input and output at the same time. This API is available in the following AWS regions: US East (N. Virginia), Europe (Frankfurt), US West (Oregon), Asia Pacific (Singapore), Asia Pacific (Sydney), Europe (London), Canada (Central), and Europe (Zurich). Visit the [documentation](#) to learn more about how to use it.
- Real-time and asynchronous speech synthesis operations.
- Newscaster speaking style is not supported in the **Generative** engine.
- Many (but not all) SSML tags are supported by Amazon Polly. For more information about NTT-supported SSML tags, see [Supported SSML tags](#)
- As with standard voices, you can choose from various sampling rates to optimize the bandwidth and audio quality for your application. Valid sampling rates for standard and neural voices are 8 kHz, 16 kHz, 22 kHz, or 24 kHz. The default for standard voices is 22 kHz. The default for generative voices is 24 kHz. Amazon Polly supports MP3, OGG (Vorbis), and raw PCM audio stream formats.

Support for generating speech marks is currently not available.

 Note

Currently Europe (London), Canada (Central) and Europe (Zurich) regions only support the following Generative voices: Joanna (en-US), Ruth (en-US), Salli (en-US), Stephen (en-US),

Tiffany (en-US), Amy (en-GB), Brian (en-GB), Olivia (en-AU), Florian (fr-FR), Ambre (fr-FR), Lorenzo (it-IT), Beatrice (it-IT), Jasmine (en-SG), Aria (en-NZ), Lennart (de-DE), Vicki (de-DE), Sabrina (de-CH), Hannah (de-AT), Niamh (en-IE), Camila (pt-BR), Lisa (nl-BE), and Seoyeon (ko-KR)

Note

In the unlikely event of model hallucination, (and with the Generative engine's model behavior of rendering the speech token by token) an imposed emergency stop mechanism is in place. The built-in mechanism stops the model from rendering speech any further. This safety feature is based on data analysis where the model has the potential to hallucinate, usually at the end of the sentence.

There could be cases where the model thinks it is going to hallucinate and then might end up cutting a word during a generation step, thus rendering half the word. This could potentially generate inappropriate results.

This safety feature reduces but does not eliminate the risk. In some edge cases, the model might continue generating random content beyond the stop threshold. Do not assume complete protection against hallucinated speech output.

Long-form voices

Amazon Polly has a **Long-form engine** that produces human-like, highly expressive, and emotionally adept voices. Long-form voices are designed to captivate listeners' attention for longer content, such as news articles, training materials, or marketing videos.

Amazon Polly Long-form voices are developed with a cutting-edge deep learning TTS technology. The model learns to replicate phonemes, prosody, intonation, and other phonetic and acoustic aspects of human language, resulting in a highly natural speech output.

The Long-form engine uses text embeddings to interpret the meaning of a text. Using text embeddings, the Long-form engine can generate the correct emphasis, pauses, and tone of a natural voice. The result is a voice that combines the complete range of emotional elements present in human communication. This includes mimicking surprisal or differentiating dialogue from narration. Together, this creates a premium speech product that sounds like a live human being.

Note

The state-of-the-art technology underlying these voices falls within the paradigm of generative AI for language and voice modelling. A side effect of the technology is that any updates to the training data and the model could result in a slight variations to the way the voices sound, even in case when their overall quality improves with model updates. This could have an impact on use cases with different content parts synthesized over a long time period – for example, a season of podcasts.

Available long-form voices

Amazon Polly currently offers four en-US and two es-ES long-form voices. Both languages have female and male voices available. The English long-form voices Daniel, Gregory, and Ruth are also available in a conversational NTTS variant.

	Language	Language code	Name/ID	Gender
1	English (US)	en-US	Danielle	Female
			Gregory	Male
			Ruth	Female
			Patrick	Male
2	Spanish (Spain)	es-ES	Alba	Female
			Raúl	Male

Feature and region compatibility

Amazon Polly long-form voices are available in the following regions:

- US East (N. Virginia): us-east-1
- Other regions not available

The Amazon Polly Long-form engine supports the following features:

- Real-time and asynchronous speech synthesis operations.
- All [speech marks](#).
- Many (but not all) SSML tags are supported by Amazon Polly. For more information about NTT-supported SSML tags, see [Supported SSML tags](#)
- As with standard voices, you can choose from various sampling rates to optimize the bandwidth and audio quality for your application. Valid sampling rates for standard, long-form, and neural voices are: 8 kHz, 16 kHz, 22kHz, or 24 kHz. The default for standard voices is 22 kHz. The default for long-form and neural voices is 24 kHz. Amazon Polly supports MP3, OGG (Vorbis), and raw PCM audio stream formats.

Note

Long-form voices cost is specified on the [Amazon Polly pricing information page](#).

Neural voices

Amazon Polly has a **Neural text-to-speech (NTTS) engine** that can produce even higher quality voices than its standard voices. Standard TTS voices use concatenative synthesis. The standard engine concatenates phonemes of recorded speech, producing very natural-sounding synthesized speech. However, the inevitable variations in speech and the techniques used to segment the waveforms limits the quality of speech. The Amazon Polly NTTS engine doesn't use standard concatenative synthesis to produce speech. It has two parts:

- A neural network — that converts a sequence of phonemes (the most basic units of language) into a sequence of *spectrograms*. (Spectrograms are snapshots of the energy levels in different frequency bands.)
- A vocoder — that converts spectrograms into a nearly continuous audio signal.

The first component of the neural TTS system is a sequence-to-sequence model. This model doesn't create its results solely from the corresponding input but also considers how the sequence of the elements of the input work together. The model chooses the spectrograms that it outputs so that their frequency bands emphasize acoustic features that the human brain uses when processing speech.

The output of this model then passes to a neural vocoder. This converts the spectrograms into speech waveforms. When trained on the large datasets used to build general-purpose concatenative-synthesis systems, this sequence-to-sequence approach will yield higher-quality, more natural-sounding voices.

Available neural voices

Neural voices are available in 36 languages and language variants. The following table lists the voices.

	Language and language variants	Language code	Name/ID	Gender
1	Arabic (Gulf)	ar-AE	Hala	Female
			Zayd	Male
2	Belgian Dutch (Flemish)	nl-BE	Lisa	Female
3	Catalan	ca-ES	Arlet	Female
4	Czech	cs-CZ	Jitka	Female
5	Chinese (Cantonese)	yue-CN	Hiujin	Female
6	Chinese (Mandarin)	cmn-CN	Zhiyu	Female
7	Danish	da-DK	Sofie	Female
8	Dutch	nl-NL	Laura	Female
9	English (Australian)	en-AU	Olivia	Female
10	English (British)	en-GB	Amy*	Female
			Emma	Female

	Language and language variants	Language code	Name/ID	Gender
			Brian	Male
			Arthur	Male
11	English (Indian)	en-IN	Kajal	Female
12	English (Irish)	en-IE	Niamh	Female
13	English (New Zealand)	en-NZ	Aria	Female
14	English (Singaporean)	en-SG	Jasmine	Female
15	English (South African)	en-ZA	Ayanda	Female

	Language and language variants	Language code	Name/ID	Gender
16	English (US)	en-US	Danielle	Female
			Gregory	Male
			Ivy	Female (child)
			Joanna*	Female
			Kendra	Female
			Kimberly	Female
			Salli	Female
			Joey	Male
			Justin	Male (child)
			Kevin	Male (child)
			Matthew*	Male
			Ruth	Female
			Stephen	Male
17	Finnish	fi-FI	Suvi	Female
18	French (Belgian)	fr-BE	Isabelle	Female
19	French (Canadian)	fr-CA	Gabrielle	Female
			Liam	Male
20	French	fr-FR	Léa	Female
			Rémi	Male

	Language and language variants	Language code	Name/ID	Gender
21	German	de-DE	Vicki	Female
			Daniel	Male
22	German (Austrian)	de-AT	Hannah	Female
23	German (Swiss)	de-CH	Sabrina	Female
24	Hindi	hi-IN	Kajal	Female
25	Italian	it-IT	Bianca	Female
			Adriano	Male
26	Japanese	ja-JP	Takumi	Male
			Kazuha	Female
			Tomoko	Female
27	Korean	ko-KR	Seoyeon	Female
			Jihye	Female
28	Norwegian	nb-NO	Ida	Female
29	Polish	pl-PL	Ola	Female
30	Portuguese (Brazilian)	pt-BR	Camila	Female
			Vitória/Vitoria	Female
			Thiago	Male
31	Portuguese (European)	pt-PT	Inês/Ines	Female

	Language and language variants	Language code	Name/ID	Gender
32	Spanish (Spain)	es-ES	Lucia	Female
			Sergio	Male
33	Spanish (Mexican)	es-MX	Mia	Female
			Andrés	Male
34	Spanish (US)	es-US	Lupe*	Female
			Pedro	Male
35	Swedish	sv-SE	Elin	Female
36	Turkish	tr-TR	Burcu	Female

*The Amy, Joanna, Lupe, and Matthew voices can be used with the Newscaster speaking style. For more information, see [Applying the newscaster voice](#).

Note

For certain voices, Amazon Polly might automatically switch to the Standard engine when accuracy is critical. For example, this can happen during letter-by-letter spelling. You might notice an audible shift in voice quality compared to Neural synthesis.

Feature and region compatibility

Neural voices aren't available in all AWS Regions, nor do they support all Amazon Polly features.

Neural voices are supported in the following regions:

- US East (N. Virginia): us-east-1
- US West (Oregon): us-west-2
- Africa (Cape Town): af-south-1

- Asia Pacific (Tokyo): ap-northeast-1
- Asia Pacific (Seoul): ap-northeast-2
- Asia Pacific (Osaka): ap-northeast-3
- Asia Pacific (Mumbai): ap-south-1
- Asia Pacific (Singapore): ap-southeast-1
- Asia Pacific (Sydney): ap-southeast-2
- Asia Pacific (Malaysia): ap-southeast-5
- Asia Pacific (Thailand): ap-southeast-7
- Canada (Central): ca-central-1
- Europe (Frankfurt): eu-central-1
- Europe (Ireland): eu-west-1
- Europe (London): eu-west-2
- Europe (Paris): eu-west-3
- Europe (Spain): eu-south-2
- Europe (Zurich): eu-central-2
- AWS GovCloud (US-West): us-gov-west-1

Endpoints and protocols for these Regions are identical to those used for standard voices. For more information, see [Amazon Polly endpoints and quotas](#).

The following features are supported for neural voices:

- Real-time and asynchronous speech synthesis operations.
- Newscaster speaking style. For more information about the speaking styles, see [Applying the newscaster voice](#).
- All speech marks.
- Many (but not all) of the SSML tags that are supported by Amazon Polly. For more information about NTTS-supported SSML tags, see Supported Tags.

As with standard voices, you can choose from various sampling rates to optimize the bandwidth and audio quality for your application. Valid sampling rates for standard and neural voices are 8

kHz, 16 kHz, 22 kHz, or 24 kHz. The default for standard voices is 22 kHz. The default for neural voices is 24 kHz. Amazon Polly supports MP3, OGG (Vorbis), and raw PCM audio stream formats.

Standard voices

Amazon Polly has a **standard** engine that use concatenative synthesis. The standard engine concatenates phonemes of recorded speech, producing very natural-sounding synthesized speech.

Available Standard voices

Amazon Polly currently offers 40 female and 20 male standard voices in 29 language and language variants.

	Language	Language code	Name/ID	Gender
1	Arabic	arb	Zeina	Female
2	Chinese (Mandarin)	cmn-CN	Zhiyu	Female
3	Danish	da-DK	Naja	Female
			Mads	Male
4	Dutch	nl-NL	Lotte	Female
			Ruben	Male
5	English (Australian)	en-AU	Nicole	Female
			Russell	Male
6	English (British)	en-GB	Amy	Female
			Emma	Female
			Brian	Male
7	English (Indian)	en-IN	Aditi	Female
			Raveena	Female

	Language	Language code	Name/ID	Gender
8	English (US)	en-US	Ivy	Female
			Joanna	Female
			Kendra	Female
			Kimberly	Female
			Salli	Female
			Joey	Male
			Kevin	Male
9	English (Welsh)	en-GB-WLS	Geraint	Male
10	French	fr-FR	Céline/Celine	Female
			Léa	Female
			Mathieu	Male
11	French (Canadian)	fr-CA	Chantal	Female
12	German	de-DE	Marlene	Female
			Vicki	Female
			Hans	Male
13	Hindi	hi-IN	Aditi	Female
14	Icelandic	is-IS	Dóra/Dora	Female
			Karl	Male

	Language	Language code	Name/ID	Gender
15	Italian	it-IT	Carla	Female
			Bianca	Female
			Giorgio	Male
16	Japanese	ja-JP	Mizuki	Female
			Takumi	Male
17	Korean	ko-KR	Seoyeon	Female
18	Norwegian	nb-NO	Liv	Female
19	Polish	pl-PL	Ewa	Female
			Maja	Female
			Jacek	Male
			Jan	Male
20	Portuguese (Brazilian)	pt-BR	Camila	Female
			Vitória/Vitoria	Female
			Ricardo	Male
21	Portuguese (European)	pt-PT	Inês/Ines	Female
			Cristiano	Male
22	Romanian	ro-RO	Carmen	Female
23	Russian	ru-RU	Tatyana	Female
			Maxim	Male

	Language	Language code	Name/ID	Gender
24	Spanish (Spain)	es-ES	Conchita	Female
			Lucia	Female
			Enrique	Male
25	Spanish (Mexican)	es-MX	Mia	Female
26	Spanish (US)	es-US	Lupe	Female
			Penélope/ Penelope	Female
			Miguel	Male
27	Swedish	sv-SE	Astrid	Female
28	Turkish	tr-TR	Filiz	Male
29	Welsh	cy-GB	Gwyneth	Female

Feature and region compatibility

Amazon Polly standard voices are available in the following Amazon Polly regions:

- US East (N. Virginia): us-east-1
- US East (Ohio): us-east-2
- US West (N. California): us-west-1
- US West (Oregon): us-west-2
- Africa (Cape Town): af-south-1
- Asia Pacific (Hong Kong): ap-east-1
- Asia Pacific (Tokyo): ap-northeast-1
- Asia Pacific (Seoul): ap-northeast-2
- Asia Pacific (Osaka): ap-northeast-3

- Asia Pacific (Mumbai): ap-south-1
- Asia Pacific (Singapore): ap-southeast-1
- Asia Pacific (Sydney): ap-southeast-2
- Asia Pacific (Malaysia): ap-southeast-5
- China (Ningxia): cn-northwest-1;
- Canada (Central): ca-central-1
- Europe (Frankfurt): eu-central-1
- Europe (Ireland): eu-west-1
- Europe (London): eu-west-2
- Europe (Paris): eu-west-3
- Europe (Spain): eu-south-2
- Europe (Stockholm): eu-north-1
- Middle East (Bahrain): me-south-1
- South America (São Paulo): sa-east-1
- AWS GovCloud (US-West): us-gov-west-1

Endpoints and protocols for these Regions are identical to those used for Neural voices. For more information, see [Amazon Polly endpoints and quotas](#).

The Amazon Polly standard engine supports the following features (TBD):

- Real-time and asynchronous speech synthesis operations.
- All [speech marks](#).
- Many (but not all) SSML tags are supported by Amazon Polly. For more information about NTTS-supported SSML tags, see [Supported SSML tags](#).
- You can choose from various sampling rates to optimize the bandwidth and audio quality for your application. The default sampling rates for standard voices are 22 kHz. Amazon Polly supports MP3, OGG (Vorbis), and raw PCM audio stream formats.

Note

Standard voices cost is specified on the [Amazon Polly pricing information page](#).

Choosing a voice engine

You can access Amazon Polly voices through the Amazon Polly console or AWS CLI.

To choose a voice engine on the console

1. Open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. From the Amazon Polly console, choose the desired voice engine.
3. Choose the desired voice from the voice drop-down menu.
4. Generate TTS audio with text of your choice.

To choose a voice engine in the AWS CLI, specify the `Engine` and `VoiceId` in the `SynthesizeSpeech` or `StartSpeechSynthesisTask` API operations. For some examples, see the [quick-start code samples](#) and the [Python examples](#).

Speech marks

Speech marks are metadata that describe the speech that you synthesize, such as where a sentence or word starts and ends in the audio stream. When you request speech marks for your text, Amazon Polly returns this metadata instead of synthesized speech. By using speech marks in conjunction with the synthesized speech audio stream, you can provide your applications with an enhanced visual experience.

For example, combining the metadata with the audio stream from your text can enable you to synchronize speech with facial animation (lip-syncing) or to highlight written words as they're spoken.

Speechmarks are available when using neural, long-form, or standard text-to-speech engines.

Topics

- [Speech mark types](#)
- [Visemes and Amazon Polly](#)
- [Speech mark output](#)
- [Requesting speech marks](#)
- [Speech marks without SSML example](#)
- [Speech marks with SSML example](#)

Speech mark types

You request speech marks using the [SpeechMarkTypes](#) option for either the [SynthesizeSpeech](#) or [StartSpeechSynthesisTask](#) commands. You specify the metadata elements that you want to return from your input text. You can request as many as four types of metadata but you must specify at least one per request. No audio output is generated with the request.

In the AWS CLI, for example:

```
--speech-mark-types='["sentence", "word", "viseme", "ssml"]'
```

Amazon Polly generates speech marks using the following elements:

- **sentence** – Indicates a sentence element in the input text.

- **word** – Indicates a word element in the text.
- **viseme** – Describes the face and mouth movements corresponding to each phoneme being spoken. For more information, see [Visemes and Amazon Polly](#).
- **ssml** – Describes a <mark> element from the SSML input text. For more information, see [Generating speech from SSML documents](#).

Visemes and Amazon Polly

A *viseme* represents the position of the face and mouth when saying a word. It is the visual equivalent of a phoneme, which is the basic acoustic unit from which a word is formed. Visemes are the basic visual building blocks of speech.

Each language has a set of viseme that correspond to their specific phonemes. In a language, each phoneme has a corresponding viseme that represents the shape that the mouth makes when forming the sound. However, not all visemes can be mapped to a particular phoneme because numerous phonemes appear the same when spoken, even though they sound different. For example, in English, the words "pet" and "bet" are acoustically different. However, when observed visually (without sound), they look exactly the same.

The following chart shows a partial list of International Phonetic Alphabet (IPA) phonemes and Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) symbols as well as their corresponding visemes for US English voices.

For the complete table and tables for all available languages, see [Languages in Amazon Polly](#).

IPA	X-SAMPA	Description	Example	Viseme
Consonants				
b	b	Voiced bilabial plosive	bed	p
d	d	Voiced alveolar plosive	dig	t
ɖʒ	dʒ	Voiced postalveolar affricate	jump	s

IPA	X-SAMPA	Description	Example	Viseme
ð	D	Voiced dental fricative	then	T
f	f	Voiceless labiodental fricative	five	f
g	g	Voiced velar plosive	game	k
h	h	Voiceless glottal fricative	house	k
...	...	...	...	...

Speech mark output

Amazon Polly returns speech mark objects in a line-delimited JSON stream. A speech mark object contains the following fields:

- **time** – the timestamp in milliseconds from the beginning of the corresponding audio stream
- **type** – the type of speech mark (sentence, word, viseme, or ssml)
- **start** – the offset in bytes (not characters) of the start of the object in the input text (not including viseme marks)
- **end** – the offset in bytes (not characters) of the object's end in the input text (not including viseme marks)
- **value** – this varies depending on the type of speech mark
 - **SSML**: <mark> SSML tag
 - **viseme**: the viseme name
 - **word** or **sentence**: a substring of the input text, as delimited by the start and end fields

For example, Amazon Polly generates the following word speech mark object from the text "Mary had a little lamb":

```
{"time":373,"type":"word","start":5,"end":8,"value":"had"}
```

The described word ("had") begins 373 milliseconds after the audio stream begins, and starts at byte 5 and ends at byte 8 of the input text.

 Note

This metadata is for the Joanna voice-id. If you use another voice with the same input text, the metadata might differ.

Requesting speech marks

You can use the console or the `synthesize-speech` command to request speech marks from Amazon Polly. You can then view the metadata or save it to a file.

Console

To generate speech marks on the console

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Text-to-Speech** tab.
3. Turn on **SSML** to use SSML.
4. Type or paste your text into the input box.
5. For **Language**, choose the language of your text.
6. For **Voice**, choose the voice you want to use.
7. To change text pronunciation, expand **Additional settings**, turn on **Customize pronunciation**, and for **Apply lexicon**, choose the desired lexicon.
8. To verify the speech, choose **Listen**.
9. Turn on **Speech file format settings**.

 Note

Downloading MP3, OGG, PCM, Mu-law, or A-law formats will not generate speech marks.

10. For **File Format**, choose **Speech marks**.
11. For **Speech mark types**, choose the types of speech marks to generate. The option to choose **SSML** metadata is only available when **SSML** is on. For more information on using SSML with Amazon Polly see [Generating speech from SSML documents](#).
12. Choose **Download**.

AWS CLI

In addition to the input text, the following elements are required to return this metadata:

- `output-format`

Amazon Polly supports only the JSON format when returning speech marks.

```
--output-format json
```

If you use an unsupported output format, Amazon Polly throws an exception.

- `voice-id`

To ensure that the metadata matches the associated audio stream, specify the same voice that is used to generate the synthesized speech audio stream. The available voices don't have identical speech rates. If you use a voice other than the one used to generate the speech, the metadata will not match the audio stream.

```
--voice-id Joanna
```

- `speech-mark-types`

Specify the type or types of speech marks you want. You can request any or all of the speech mark types, but must specify at least one type.

```
--speech-mark-types='["sentence", "word", "viseme", "ssml"]'
```

- `text-type`

Plain text is the default input text for Amazon Polly, so you must use `text-type ssml` if you want to return SSML speech marks.

- `outfile`

Specify the output file to which the metadata is written.

```
MaryLamb.txt
```

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly synthesize-speech \  
  --output-format json \  
  --voice-id Voice ID \  
  --text 'Input text' \  
  --speech-mark-types='["sentence", "word", "viseme"]' \  
outfile
```

Speech marks without SSML example

The following example shows you what requested metadata looks like on your screen for the simple sentence: "Mary had a little lamb." For simplicity, we don't include SSML speech marks in this example.

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly synthesize-speech \  
  --output-format json \  
  --voice-id Joanna \  
  --text 'Mary had a little lamb.' \  
  --speech-mark-types='["viseme", "word", "sentence"]' \  
MaryLamb.txt
```

When you make this request, Amazon Polly returns the following in the .txt file:

```
{"time":0,"type":"sentence","start":0,"end":23,"value":"Mary had a little lamb."}  
{"time":6,"type":"word","start":0,"end":4,"value":"Mary"}  
{"time":6,"type":"viseme","value":"p"}
```

```
{
  "time": 73, "type": "viseme", "value": "E"
}, {
  "time": 180, "type": "viseme", "value": "r"
}, {
  "time": 292, "type": "viseme", "value": "i"
}, {
  "time": 373, "type": "word", "start": 5, "end": 8, "value": "had"
}, {
  "time": 373, "type": "viseme", "value": "k"
}, {
  "time": 460, "type": "viseme", "value": "a"
}, {
  "time": 521, "type": "viseme", "value": "t"
}, {
  "time": 604, "type": "word", "start": 9, "end": 10, "value": "a"
}, {
  "time": 604, "type": "viseme", "value": "@"
}, {
  "time": 643, "type": "word", "start": 11, "end": 17, "value": "little"
}, {
  "time": 643, "type": "viseme", "value": "t"
}, {
  "time": 739, "type": "viseme", "value": "i"
}, {
  "time": 769, "type": "viseme", "value": "t"
}, {
  "time": 799, "type": "viseme", "value": "t"
}, {
  "time": 882, "type": "word", "start": 18, "end": 22, "value": "lamb"
}, {
  "time": 882, "type": "viseme", "value": "t"
}, {
  "time": 964, "type": "viseme", "value": "a"
}, {
  "time": 1082, "type": "viseme", "value": "p"
}
```

In this output, each part of the text is broken out in terms of speech marks:

- The sentence "Mary had a little lamb."
- Each word in the text: "Mary", "had", "a", "little", and "lamb."
- The viseme for each sound in the corresponding audio stream: "p", "E", "r", "i", and so on. For more information on visemes see [Visemes and Amazon Polly](#).

Speech marks with SSML example

The process of generating speech marks from SSML-enhanced text is similar to the process when SSML is not present. Use the `synthesize-speech` command, and specify the SSML-enhanced text and the type of speech marks that you want, as shown in the following example. To make the example easier to read, we don't include viseme speech marks, but these could be included as well.

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly synthesize-speech \
  --output-format json \
  --voice-id Joanna \
```

```
--text-type ssml \  
--text '<speak><prosody volume="+20dB">Mary had <break time="300ms"/>a little <mark  
name="animal"/>lamb</prosody></speak>' \  
--speech-mark-types='["sentence", "word", "ssml"]' \  
output.txt
```

When you make this request, Amazon Polly returns the following in the .txt file:

```
{"time":0,"type":"sentence","start":31,"end":95,"value":"Mary had <break time=\"300ms  
\"/>a little <mark name=\"animal\"/>lamb"}  
{"time":6,"type":"word","start":31,"end":35,"value":"Mary"}  
{"time":325,"type":"word","start":36,"end":39,"value":"had"}  
{"time":897,"type":"word","start":40,"end":61,"value":"<break time=\"300ms\"/>"}  
{"time":1291,"type":"word","start":61,"end":62,"value":"a"}  
{"time":1373,"type":"word","start":63,"end":69,"value":"little"}  
{"time":1635,"type":"ssml","start":70,"end":91,"value":"animal"}  
{"time":1635,"type":"word","start":91,"end":95,"value":"lamb"}
```

Generating speech from SSML documents

You can use Amazon Polly to generate speech from either plain text or from documents marked up with Speech Synthesis Markup Language (SSML). Using SSML-enhanced text gives you additional control over how Amazon Polly generates speech from the text you provide.

With SSML tags, you can customize and control aspects of speech such as pronunciation, volume, and speech rate. In the AWS Management Console, the SSML-enhanced text that you want to convert to audio is entered on the SSML tab of the Text-to-Speech page. Although text entered in plain text relies on default settings for the language and voice you've chosen, text enhanced with SSML tells Amazon Polly not only what you want to say, but how you want to say it. Except for the added SSML tags, Amazon Polly synthesizes SSML-enhanced text in the same way as it synthesizes plain text. See [Synthesizing speech with Amazon Polly example](#) for more information.

When using SSML, you enclose the entire text in a `<say>` tag to let Amazon Polly know that you're using SSML. For example:

```
<say>Hi! My name is Joanna. I will read any text you type here.</say>
```

You then use specific SSML tags on the text inside the `<say>` tags to customize the way you want the text to sound. You can add a pause, change the pace of the speech, lower or raise the volume of the voice, or add many other customizations so that the text sounds right for you. For a full list of the SSML tags that you can use, see [Supported SSML tags](#).

For example, you can include a long pause within your text, or change the speech rate or pitch. Other options include:

- emphasizing specific words or phrases
- using phonetic pronunciation
- including breathing sounds
- whispering
- using the Newscaster speaking style.

For complete details on the SSML tags supported by Amazon Polly and how to use them, see [Supported SSML tags](#)

When using SSML, there are several reserved characters that require special treatment. This is because SSML uses these characters as part of its code. In order to use them, you use a specific entity to *escape* them. For more information, see [Reserved characters in SSML](#)

Amazon Polly provides these types of control with a subset of the SSML markup tags that are defined by [Speech Synthesis Markup Language \(SSML\) Version 1.1, W3C Recommendation](#).

You can use SSML within the Amazon Polly console or by using the AWS CLI. The following topics show you how you can use SSML to generate speech and control the output so that it precisely fits your needs.

Topics

- [Reserved characters in SSML](#)
- [Using SSML on the console](#)
- [Using SSML with the Synthesize-Speech command](#)
- [Synthesizing an SSML-enhanced document](#)
- [Supported SSML tags](#)

Reserved characters in SSML

There are five predefined characters that can't normally be used within an SSML statement. These entities are reserved by the language specification. These characters are as follows:

Escape
code

"
quotation
mark
(double
quotation
mark)

&
ampersand

'
apostroph
e
or

Escape code

single
quotation
mark

<
less
than
sign

>
greater
than
sign

Because SSML uses these characters as part of its code, to use these symbols in SSML, you must *escape* the character when you use it. You use the escape code instead of the actual character so it displays properly while still creating a valid SSML document. For example, the following sentence

We're using the lawyer at Peabody & Chambers, attorneys-at-law.

would be rendered in SSML as

```
<speaK>
We&apos;re using the lawyer at Peabody &amp; Chambers, attorneys-at-law.
</speaK>
```

In this case, the special characters for the apostrophe and ampersand are escaped so the SSML document remains valid.

For the **&**, **<**, and **>** symbols, escape codes are always necessary when you use SSML. Additionally, when you use the apostrophe/single quotation mark (') as an apostrophe, you must also use the escape code.

However, when you use the double quotation mark ("), or the apostrophe/single quotation mark (') as a quotation mark, then whether or not you use the escape code is dependent on context.

Double quotation marks

- Must be escaped when in a attribute value delimited by double quotes. For example, in the following AWS CLI code

```
--text "Pete &quot;Maverick&quot; Mitchell"
```

- Do not need to be escaped when in textual context. For example, in the following

```
He said, "Turn right at the corner."
```

- Do not need to be escaped when in a attribute value delimited by single quotes. For example, in the following AWS CLI code

```
--text 'Pete "Maverick" Mitchell'
```

Single quotation marks

- Must be escaped when used as an apostrophe. For example, in the following

```
We&apos;ve got to leave quickly.
```

- Do not need to be escaped when in textual context. For example, in the following

```
"And then I said, 'Don't quote me.'"
```

- Do not need to be escaped when in a code attribute delimited by double quotes. For example, in the following AWS CLI code

```
--text "Pete 'Maverick' Mitchell"
```

Using SSML on the console

In the following example, you use an SSML tag to tell Amazon Polly to substitute "World Wide Web Consortium" for "W3C" when it speaks a short paragraph. You also use tags to introduce a pause and whisper a word. Compare the results of this exercise with that of [Applying lexicons \(Synthesizing Speech\)](#).

For more information on SSML, with examples, see [Supported SSML tags](#).

To synthesize speech from SSML-enhanced text (console)

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. If it isn't already displayed, choose the **Text-to-Speech** tab.
3. Turn on **SSML**.
4. Type or paste the following text in the text box:

```
< speak >
  He was caught up in the game.< break time="1s"/> In the middle of the
  10/3/2014 < sub alias="World Wide Web Consortium">W3C</sub> meeting,
  he shouted, "Nice job!" quite loudly. When his boss stared at him, he
  repeated
  < amazon:effect name="whispered">"Nice job,"</amazon:effect> in a
  whisper.
</ speak >
```

The SSML tags tell Amazon Polly how to render the text:

- `< break time="1s"/>` tells Amazon Polly to pause 1 second between the first two sentences.
- `< sub alias="World Wide Web Consortium">W3C</sub>` tells Amazon Polly to substitute World Wide Web Consortium for the acronym W3C.
- `< amazon:effect name="whispered">Nice job</amazon:effect>` tells Amazon Polly to whisper the second instance of "Nice job." .

Note

When you use the AWS CLI, you enclose the input text in quotation marks to differentiate it from the surrounding code. The Amazon Polly console doesn't show you code, so you don't enclose input text in quotation marks when you use it.

5. For **Language**, choose **English, US**, then choose a voice.
6. To listen to the speech, choose **Listen**.
7. To save the speech file, choose **Download**. If you want to save it in a different format, expand **Additional settings**, turn on **Speech file format settings** and choose the format that you want, then choose **Download**.

Using SSML with the Synthesize-Speech command

This example shows how to use the `synthesize-speech` command with an SSML string. When you use the `synthesize-speech` command, you typically provide the following:

- The input text (required)
- Opening and closing tags (required)
- The output format
- A voice

In this example, you specify a simple text string in quotation marks along with the required opening and closing `<say></say>` tags.

Important

Although you don't use quotation marks around input text in the Amazon Polly console, you must use them in use the AWS CLI. It's also important that you differentiate between the quotation marks around input text and quotations required for individual tags. For example, you can use standard quotation marks (") to enclose the input text, and single quotation marks (') for interior tags, or vice versa. Either option works for Unix, Linux, and macOS. However, with Windows you must enclose the input text in standard quotations marks and use single quotation marks for the tags. For all operating systems, you can use standard quotation marks (") to enclose the input text, and single quotation marks (') for interior tags). For example:

```
--text "<say>Hello <break time='300ms'> World</say>"
```

For Unix, Linux, and macOS, you can also use the reverse, with single quotation marks (') enclosing the input text and standard quotation marks (") for interior tags:

```
--text '<say>Hello <break time="300ms"> World</say>'
```

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly synthesize-speech \  
--text-type ssml \  
--text '<speak>Hello world</speak>' \  
--output-format mp3 \  
--voice-id Joanna \  
speech.mp3
```

To hear the synthesized speech, play the resulting `speech.mp3` file using any audio player.

Synthesizing an SSML-enhanced document

For longer input text, you may find it easier to save your SSML content to a file and simply specify the file name in the `synthesize-speech` command. For example you could save the following to a file called `example.xml`:

```
<?xml version="1.0"?>  
<speak version="1.1"  
  xmlns="http://www.w3.org/2001/10/synthesis"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://www.w3.org/2001/10/synthesis http://www.w3.org/TR/  
speech-synthesis11/synthesis.xsd"  
  xml:lang="en-US">Hello World</speak>
```

The `xml:lang` attribute specifies en-US (US English) as the language of the input text. For information about how the language of the input text and the language of the chosen voice affect the `SynthesizeSpeech` operation, see [Specifying another language for specific words](#).

To run an SSML-enhanced file

1. Save the SSML to a file (for example, `example.xml`).
2. Run the following `synthesize-speech` command from the path where the XML file is stored and specify the SSML file as input by substituting `file:\\example.xml` for the input text. Because this command points to a file instead of containing the actual input text, you don't use quotation marks.

Note

The following AWS CLI 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 polly synthesize-speech \
--text-type ssml \
--text file://example.xml \
--output-format mp3 \
--voice-id Joanna \
speech.mp3
```

3. To hear the synthesized speech, play the resulting `speech.mp3` file using any audio player.

Supported SSML tags

All tags except for `<amazon:domain name="news">` are supported for Standard voices. Tag availability for other voices is provided in the following table.

Amazon Polly supports the following SSML tags:

Action	SSML tag	Neural voice availability	Long-form voice availability	Generative voice availability
Adding a pause	<code><break></code>	Full availability	Full availability	Full availability
Emphasizing words	<code><emphasis></code>	Not available	Not available	Not available
Specifying another language for specific words	<code><lang></code>	Full availability	Full availability	Full availability

Action	SSML tag	Neural voice availability	Long-form voice availability	Generative voice availability
Placing a custom tag in your text	<mark>	Full availability	Full availability	Partial availability
Adding a pause between paragraphs	<p>	Full availability	Full availability	Full availability
Using phonetic pronunciation	<phoneme>	Full availability	Full availability	Partial availability
Controlling volume, speaking rate, and pitch	<prosody>	Partial availability	Partial availability	Partial availability
Setting a maximum duration for synthesized speech	<prosody amazon:max-duration>	Not available	Not available	Not available
Adding a pause between sentences	<s>	Full availability	Full availability	Full availability
Controlling how special types of words are spoken	<say-as>	Partial availability	Full availability	Full availability
Identifying SSML-enhanced text	<speak>	Full availability	Full availability	Full availability
Pronouncing acronyms and abbreviations	<sub>	Full availability	Full availability	Full availability
Improving pronunciation by specifying parts of speech	<w>	Full availability	Full availability	Full availability

Action	SSML tag	Neural voice availability	Long-form voice availability	Generative voice availability
Adding the sound of breathing	<amazon:auto-breaths>	Not available	Not available	Not available
Newscaster speaking style	<amazon:domain name="news">	Select neural voices only	Not available	Not available
Adding dynamic range compression	<amazon:effect name="drc">	Full availability	Full availability	Not available
Speaking softly	<amazon:effect phonation="soft">	Not available	Not available	Not available
Controlling timbre	<amazon:effect vocal-tract-length>	Not available	Not available	Not available
Whispering	<amazon:effect name="whispered">	Not available	Not available	Not available

Note

If you use unsupported SSML tags in standard, neural, or long-form format, you will get an error.

Identifying SSML-enhanced text

<spea<

This tag is supported by generative, long-form, neural, and standard TTS formats.

The <spea< tag is the root element of all Amazon Polly SSML text. All SSML-enhanced text must be enclosed within a pair of <spea< tags.


```
<speake>Mary had a little lamb.</speake>
```

Adding a pause

<break>

This tag is supported by generative, long-form, neural, and standard TTS formats.

To add a pause to your text, use the `<break>` tag. You can set a pause based on strength (equivalent to the pause after a comma, a sentence, or a paragraph), or you can set it to a specific length of time in seconds or milliseconds. If you don't specify an attribute to determine the pause length, Amazon Polly uses the default, which is `<break strength="medium"/>`, which adds a pause the length of a pause after a comma.

strength attribute values:

- `none`: No pause. Use `none` to remove a normally occurring pause, such as after a period.
- `x-weak`: Has the same strength as `none`, no pause.
- `weak`: Sets a pause of the same duration as the pause after a comma.
- `medium`: Has the same strength as `weak`.
- `strong`: Sets a pause of the same duration as the pause after a sentence.
- `x-strong`: Sets a pause of the same duration as the pause after a paragraph.

time attribute values:

- `[number]s`: The duration of the pause, in seconds. The maximum duration is 10s.
- `[number]ms`: The duration of the pause, in milliseconds. The maximum duration is 10000ms.

For example:

```
<speake>  
    Mary had a little lamb <break time="3s"/>Whose fleece was white as snow.  
</speake>
```

If you don't use an attribute with the `break` tag, the result varies depending on text:

- If there is no other punctuation next to the break tag, it creates a `<break strength="medium"/>` (comma-length pause).
- If the tag is next to a comma, it upgrades the tag to a `<break strength="strong"/>` (sentence-length pause).
- If the tag is next to a period, it upgrades the tag to `<break strength="x-strong"/>` (paragraph-length pause).

Emphasizing words

`<emphasis>`

This tag is supported only by the standard TTS format.

To emphasize words, use the `<emphasis>` tag. Emphasizing words changes the speaking rate and volume. More emphasis makes Amazon Polly speak the text louder and slower. Less emphasis makes it speak quieter and faster. To specify the degree of emphasis, use the `level` attribute.

`level` attribute values:

- **Strong:** Increases the volume and slows the speaking rate so that the speech is louder and slower.
- **Moderate:** Increases the volume and slows the speaking rate, but less than strong. Moderate is the default.
- **Reduced:** Decreases the volume and speeds up the speaking rate. Speech is softer and faster.

Note

The normal speaking rate and volume for a voice falls between the moderate and reduced levels.

For example:

```
<speaking>I already told you I <emphasis level="strong">really like</emphasis> that  
person.</speaking>
```

Specifying another language for specific words

`<lang>`

This tag is supported by generative, long-form, neural, and standard TTS formats. For generative voices, the `<lang>` tag can be used only around full sentences.

Specify another language for a specific word, phrase, or sentence with the `<lang>` tag. Foreign language words and phrases are generally spoken better when they are enclosed within a pair of `<lang>` tags. To specify the language, use the `xml:lang` attribute. For a complete list of available languages, see [Languages in Amazon Polly](#).

Unless you apply the `<lang>` tag, all of the words in the input text are spoken in the language of the voice specified in the `voice-id`. If you apply the `<lang>` tag, the words are spoken in that language.

For example, if the `voice-id` is Joanna (who speaks US English), Amazon Polly speaks the following in the Joanna voice without a French accent:

```
<speak>
    Je ne parle pas français.
</speak>
```

If you use the Joanna voice with the `<lang>` tag, Amazon Polly speaks the sentence in the Joanna voice in American-accented French:

```
<speak>
    <lang xml:lang="fr-FR">Je ne parle pas français.</lang>.
</speak>
```

Because Joanna is not a native French voice, pronunciation is based on her native language, US English. For example, although perfect French pronunciation features an uvular trill /R/ in the word *français*, Joanna's US English voice pronounces this phoneme as the corresponding sound /r/.

If you use the `voice-id` of Giorgio, who speaks Italian, with the following text, Amazon Polly speaks the sentence in Giorgio's voice with an Italian pronunciation:

```
< speak>
  Mi piace Bruce Springsteen.
</ speak>
```

If you use the same voice with the following `<lang>` tag, Amazon Polly pronounces Bruce Springsteen in Italian-accented English:

```
< speak>
  Mi piace < lang xml:lang="en-US">Bruce Springsteen.</ lang>
</ speak>
```

This tag can also be used as a substitute for the optional [DefaultLangCode](#) option when synthesizing speech. However, doing so requires that you format your text using SSML.

Placing a custom tag in your text

`< mark>`

This tag is supported by long-form, neural, and standard TTS formats. This tag does not do anything for generative voices because speechmarks are not available for generative voices.

To put a custom tag within the text, use the `< mark>` tag. Amazon Polly takes no action on the tag, but returns the location of the tag in the SSML metadata. This tag can be anything you want to call out, as long as it maintains the following format:

```
< mark name="tag_name"/>
```

For example, suppose that the tag name is "animal" and the input text is:

```
< speak>
  Mary had a little < mark name="animal"/>lamb.
</ speak>
```

Amazon Polly might return the following SSML metadata:

```
{"time":767,"type":"ssml","start":25,"end":46,"value":"animal"}
```

Adding a pause between paragraphs

`<p>`

This tag is supported by generative, long-form, neural, and standard TTS formats.

To add a pause between paragraphs in your text, use the `<p>` tag. Using this tag provides a longer pause than native speakers usually place at commas or the end of a sentence. Use the `<p>` tag to enclose the paragraph:

```
<speack>
  <p>This is the first paragraph. There should be a pause after this text is
  spoken.</p>
  <p>This is the second paragraph.</p>
</speack>
```

This is equivalent to specifying a pause using `<break strength="x-strong"/>`.

Using phonetic pronunciation

`<phoneme>`

This tag is supported by long-form, neural, and standard TTS formats.

To make Amazon Polly use phonetic pronunciation for specific text, use the `<phoneme>` tag.

Two attributes are required with the `<phoneme>` tag. They indicate the phonetic alphabet Amazon Polly uses and the phonetic symbols of the corrected pronunciation:

- `alphabet`
 - `ipa`— Indicates that the International Phonetic Alphabet (IPA) will be used.
 - `x-sampa`— Indicates that the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) will be used.
- `ph`
 - Specifies the phonetic symbols for pronunciation. For more information, see [Languages in Amazon Polly](#)

With the `<phoneme>` tag, Amazon Polly uses the pronunciation specified by the `ph` attribute instead of the standard pronunciation associated by default with the language used by the selected voice.

For instance, the word "pecan" can be pronounced two ways. In the following example, "pecan" is assigned a different pronunciation in each line. Amazon Polly pronounces pecan as specified in the `ph` attributes, instead of using the default pronunciation.

International Phonetic Alphabet (IPA)

```
<speack>
    You say, <phoneme alphabet="ipa" ph="p##k##n">pecan</phoneme>.
    I say, <phoneme alphabet="ipa" ph="#pi.kæn">pecan</phoneme>.
</speack>
```

Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA)

```
<speack>
    You say, <phoneme alphabet='x-sampa' ph='pI"kA:n'>pecan</phoneme>.
    I say, <phoneme alphabet='x-sampa' ph='"pi.k{n'>pecan</phoneme>.
</speack>
```

Mandarin Chinese uses Pinyin for phonetic pronunciation..

Pinyin

```
<speack>
    ## <phoneme alphabet="x-amazon-pinyin" ph="bo2">#</phoneme>#
    ## <phoneme alphabet="x-amazon-pinyin" ph="bao2">#</phoneme>#
</speack>
```

Japanese uses Yomigana and Pronunciation Kana.

Yomigana

```
<speack>
    ###<phoneme alphabet="x-amazon-yomigana" ph="####">##</phoneme>###
    ###<phoneme alphabet="x-amazon-yomigana" ph="####">##</phoneme>###
    ###<phoneme alphabet="x-amazon-yomigana" ph="Hirokazu">##</phoneme>###
</speack>
```

Pronunciation Kana

```
<speak>
  ###<phoneme alphabet="x-amazon-pron-kana" ph="##'##">##</phoneme>###
</speak>
```

Controlling volume, speaking rate, and pitch

<prosody>

Prosody tag attributes are fully supported by the standard TTS voices. Generative, Neural, and Long-Form voices support the volume and rate attributes, but don't support the pitch attribute. For Generative voices, the prosody tag can be used only around full sentences.

To control the volume, rate, or pitch of your selected voice, use the *prosody* tag.

Volume, speech rate, and pitch are dependent on the specific voice selected. In addition to differences between voices for different languages, there are differences between individual voices speaking the same language. Because of this, while attributes are similar across all languages, there are clear variations from language to language and no absolute value is available.

The *prosody* tag has three attributes, each of which has several available values to set the attribute. Each attribute uses the same syntax:

```
<prosody attribute="value"></prosody>
```

- **volume**
 - **default**: Resets volume to the default level for the current voice.
 - **silent**, **x-soft**, **soft**, **medium**, **loud**, **x-loud**: Sets the volume to a predefined value for the current voice.
 - **+ndB**, **-ndB**: Changes volume relative to the current level. A value of **+0dB** means no change, **+6dB** means approximately twice the current volume, and **-6dB** means approximately half the current volume.

For example, you could set the volume for a passage as follows:

```
<speak>
  Sometimes it can be useful to <prosody volume="loud">increase the volume
```

```
    for a specific speech.</prosody>
</speak>
```

Or you could set it this way:

```
<speak>
    And sometimes a lower volume <prosody volume="-6dB">is a more effective way of
    interacting with your audience.</prosody>
</speak>
```

- **rate**
 - **x-slow, slow, medium, fast, x-fast.** Sets the pitch to a predefined value for the selected voice.
 - **n%:** A non-negative percentage change in the speaking rate. For example, a value of 100% means no change in speaking rate, a value of 200% means a speaking rate twice the default rate, and a value of 50% means a speaking rate of half the default rate. This value has a range of 20-200%.

For example, you could set the speech rate for a passage as follows:

```
<speak>
    For dramatic purposes, you might wish to <prosody rate="slow">slow up the
    speaking
    rate of your text.</prosody>
</speak>
```

Or you could set it this way:

```
<speak>
    Although in some cases, it might help your audience to <prosody rate="85%">slow
    the speaking rate slightly to aid in comprehension.</prosody>
</speak>
```

- **pitch**
 - **default:** Resets pitch to the default level for the current voice.
 - **x-low, low, medium, high, x-high:** Sets the pitch to a predefined value for the current voice.
 - **+n% or -n%:** Adjusts pitch by a relative percentage. For example, a value of +0% means no baseline pitch change, +5% gives a little higher baseline pitch, and -5% results in a little lower baseline pitch.

For example, you could set the pitch for a passage as follows:

```
<say>
  Do you like synthesized speech <prosody pitch="high">with a pitch that is higher
  than normal?</prosody>
</say>
```

Or you could set it this way:

```
<say>
  Or do you prefer your speech <prosody pitch="-10%">with a somewhat lower pitch?
</prosody>
</say>
```

The `<prosody>` tag must contain at least one attribute, but can include more within the same tag.

```
<say>
  Each morning when I wake up, <prosody volume="loud" rate="x-slow">I speak
  quite slowly and deliberately until I have my coffee.</prosody>
</say>
```

It can also be combined with nested tags, as follows:

```
<say>
  <prosody rate="85%">Sometimes combining attributes <prosody pitch="-10%">can
  change the impression your audience has of a voice</prosody> as well.</prosody>
</say>
```

Note

Currently `<prosody>` is partially available for the Generative voices.

Setting a maximum duration for synthesized speech

`<prosody amazon:max-duration>`

This tag is currently supported only by the standard TTS format.

To control how long you want a speech to take when it is synthesized, use the `<prosody>` tag with the `amazon:max-duration` attribute.

The duration of synthesized speech varies slightly, depending on the voice you select. This can make it difficult to match synthesized speech with visuals or other activities that require precise timing. This issue is magnified for translation applications because the time it takes to say particular phrases can vary widely with different languages.

The `<prosody amazon:max-duration>` tag matches synthesized speech to the amount of time you want it to take (the duration).

This tag uses the following syntax:

```
<prosody amazon:max-duration="time duration">
```

With the `<prosody amazon:max-duration>` tag, you can specify duration in either seconds or milliseconds:

- *ns*: the maximum duration in seconds
- *nms*: the maximum duration in milliseconds

For example, the following spoken text has a maximum duration of 2 seconds:

```
<speake>  
  <prosody amazon:max-duration="2s">  
    Human speech is a powerful way to communicate.  
  </prosody>  
</speake>
```

Text placed within the tag, it doesn't exceed the specified duration. If the chosen voice or language would normally take longer than that duration, Amazon Polly speeds up the speech so that it fits into the specified duration.

If the specified duration is longer than it takes to read the text at a normal rate, Amazon Polly reads the speech normally. It doesn't slow down the speech or add silence, so the resulting audio is shorter than requested.

Note

Amazon Polly increases the speed no more than 5 times the normal rate. If text is spoken faster than this, it usually doesn't make sense. If a speech cannot fit within your specified duration even when speeded up to the maximum, the audio will be speeded up but will last longer than the specified duration.

You can include a single sentence or multiple sentences within a `<prosody amazon:max-duration>` tag, and you can use multiple `<prosody amazon:max-duration>` tags within your text.

For example:

```
<say>
  <prosody amazon:max-duration="2400ms">
    Human speech is a powerful way to communicate.
  </prosody>
  <break strength="strong"/>
  <prosody amazon:max-duration="5100ms">
    Even a simple 'Hello' can convey a lot of information depending on the pitch,
    intonation, and tempo.
  </prosody>
  <break strength="strong"/>
  <prosody amazon:max-duration="8900ms">
    We naturally understand this information, which is why speech is ideal for
    creating applications where
    a screen isn't practical or possible, or simply isn't convenient.
  </prosody>
</say>
```

Using the `<prosody amazon:max-duration>` tag can increase latency when Amazon Polly returns synthesized speech. The degree of latency depends on the passage and its length. We recommend using text comprised of relatively short text passages.

Limitations

There are limitations both in how you use `<prosody amazon:max-duration>` tag and in how it works with other SSML tags:

- The text inside a `<prosody amazon:max-duration>` tag can't be longer than 1500 characters.
- You can't nest `<prosody amazon:max-duration>` tags. If you put one `<prosody amazon:max-duration>` tag inside another, Amazon Polly ignores the inner tag.

For example, in the following, the `<prosody amazon:max-duration="5s">` tag is ignored:

```
<say>
  <prosody amazon:max-duration="16s">
    Human speech is a powerful way to communicate.

    <prosody amazon:max-duration="5s">
      Even a simple 'Hello' can convey a lot of information depending on the
      pitch, intonation, and tempo.
    </prosody>

    We naturally understand this information, which is why speech is ideal for
    creating applications where a screen isn't practical or possible, or simply isn't
    convenient.
  </prosody>
</say>
```

- You can't use the `<prosody>` tags with the `rate` attribute within a `<prosody amazon:max-duration>` tag. This is because both affect the speed at which text is spoken.

In the following example, Amazon Polly ignores the `<prosody rate="2">` tag:

```
<say>
  <prosody amazon:max-duration="7500ms">
    Human speech is a powerful way to communicate.

    <prosody rate="2">
      Even a simple 'Hello' can convey a lot of information depending on the
      pitch, intonation, and tempo.
    </prosody>
  </prosody>
</say>
```

Pauses and max-duration

When using `max-duration` tag, you can still insert pauses within your text. However, Amazon Polly includes the length of the pause when calculating the maximum duration for speech.

Additionally, Amazon Polly preserves the short pauses that occur where commas and periods are placed within a passage and includes in the maximum duration.

For example, in the following block, the 600 millisecond break and the breaks caused by the commas and periods occur within the 8-second speech:

```
<speak>
  <prosody amazon:max-duration="8s">
    Human speech is a powerful way to communicate.
    <break time="600ms"/>
    Even a simple 'Hello' can convey a lot of information depending on the pitch,
    intonation, and tempo.
  </prosody>
</speak>
```

Adding a pause between sentences

<s>

This tag is supported by generative, long-form, neural, and standard TTS formats.

To add a pause between lines or sentences in your text, use the <s> tag. Using this tag has the same effect as:

- Ending a sentence with a period (.)
- Specifying a pause with <break strength="strong"/>

Unlike the <break> tag, the <s> tag encloses the sentence. This is useful for synthesizing speech that is organized in lines, rather than sentence, such as poetry.

In the following example, the <s> tag creates a short pause after both the first and second sentences. The final sentence has no <s> tag, but it is also followed by a short pause because it ends with a period.

```
<speak>
  <s>Mary had a little lamb</s>
  <s>Whose fleece was white as snow</s>
  And everywhere that Mary went, the lamb was sure to go.
```

```
</speak>
```

Controlling how special types of words are spoken

`<say-as>`

The `<say-as>` tag is supported by generative, long-form, neural, and standard TTS engines. Note, however, that if Amazon Polly is using a neural voice and encounters the `<say-as>` tag with the `characters` option at runtime, the affected sentence will be synthesized using the related standard voice. However, the affected sentence will still be billed as if it uses a neural voice.

Use the `<say-as>` tag with the `interpret-as` attribute to tell Amazon Polly how to say certain characters, words, and numbers. This enables you to provide additional context to eliminate any ambiguity on how Amazon Polly should render the text.

The `<say-as>` tag uses one attribute, `interpret-as`, which uses a number of possible available values. Each uses the same syntax:

```
<say-as interpret-as="value">[text to be interpreted]</say-as>
```

The following values are available with `interpret-as`:

- `characters` or `spell-out`: Spells out each letter of the text, as in a-b-c.

Note

This option is not currently supported for neural voices. If you're using a neural voice and this SSML code is encountered by Amazon Polly at run-time, the affected sentence will be synthesized using the related standard voice. Please note, however, that this sentence will still be billed as if it uses a neural voice.

- `cardinal` or `number`: Interprets the numerical text as a cardinal number, as in 1,234.
- `ordinal`: Interprets the numerical text as an ordinal number, as in 1,234th.
- `digits`: Spells out each digit individually, as in 1-2-3-4.
- `fraction`: Interprets the numerical text as a fraction. This works for both common fractions such as 3/20, and mixed fractions, such as 2 ½. See below for more information.

- **unit:** Interprets a numerical text as a measurement. The value should be either a number or a fraction followed by a unit with no space in between as in `1/2inch`, or by just a unit, as in `1meter`.
- **date:** Interprets the text as a date. The format of the date must be specified with the `format` attribute. See below for more information.
- **time:** Interprets the numerical text as duration, in minutes and seconds, as in `1 ' 21"`.
- **address:** Interprets the text as part of a street address.
- **expletive:** "Beeps out" the content included within the tag.
- **telephone:** Interprets the numerical text as a 7-digit or 10-digit telephone number, as in `2025551212`. You can also use this value for handle telephone extensions, as in `2025551212x345`. See below for more information.

Note

Currently the `telephone` option is not available for all languages. However, it is available for voices speaking English language variants (en-AU, en-GB, en-IN, en-US, and en-GB-WLS), Spanish language variants (es-ES, es-MX, and es-US), French language variants (fr-FR and fr-CA), and Portuguese variants (pt-BR and pt-PT), as well as German (de-DE), Italian (it-IT), Japanese (ja-JP), and Russian (ru-RU). It should also be noted that in some cases, languages such as Arabic (arb) automatically handle the number set as a telephone number and so don't actually implement the `telephone` SSML tag.

Fractions

Amazon Polly interprets values within the `say-as` tag that have the `interpret-as="fraction"` attribute as common fractions. The following is the syntax for fractions:

- *Fraction*

Syntax: *cardinal number*/*cardinal number*, such as `2/9`.

For example: `<say-as interpret-as="fraction">2/9</say-as>` is pronounced "two ninths."

- *Non-negative Mixed Number*

Syntax: *cardinal number*+*cardinal number*/*cardinal number*, such as `3+1/2`.

For example, `<say-as interpret-as="fraction">3+1/2</say-as>` is pronounced "three and a half."

Note

There must be a + between the "3" and the "1/2". Amazon Polly doesn't support a mixed number without the +, such as "3 1/2".

Dates

When `interpret-as` is set to `date`, you also need to indicate the format of the date.

This uses the following syntax:

```
<say-as interpret-as="date" format="format">[date]</say-as>
```

For example:

```
<say>  
  I was born on <say-as interpret-as="date" format="mdy">12-31-1900</say-as>.  
</say>
```

The following formats can be used with the `date` attribute.

- `mdy`: Month-day-year.
- `dmy`: Day-month-year.
- `ymd`: Year-month-day.
- `md`: Month-day.
- `dm`: Day-month.
- `ym`: Year-month.
- `my`: Month-year.
- `d`: Day.
- `m`: Month.
- `y`: Year.

- `yyyymmdd`: Year-month-day. If you use this format, you can make Amazon Polly skip parts of the date using question marks.

For example, Amazon Polly renders the following as "September 22nd":

```
<say-as interpret-as="date">????0922</say-as>
```

Format is not needed.

Telephone

Amazon Polly attempts to interpret the text you provide correctly based on the text's formatting even without the `<say-as>` tag. For example, if your text includes "202-555-1212," Amazon Polly interprets it as a 10-digit telephone number and says each digit individually, with a brief pause for each dash. In this case, you don't need to use `<say-as interpret-as="telephone">`. However, if you provide the text "2025551212" and want Amazon Polly to say it as a phone number, you would specify `<say-as interpret-as="telephone">`.

The logic for interpreting each element is language-specific. For example, US and UK English differ in how phone numbers are pronounced (in UK English, sequences of the same digit are grouped together, as in "double five" or "triple four"). To see the difference, test the following example with a US voice and with a UK voice:

```
<say-as>  
  Richard's number is <say-as interpret-as="telephone">2122241555</say-as>  
</say-as>
```

Pronouncing acronyms and abbreviations

`<sub>`

This tag is supported by generative, long-form, neural, and standard TTS formats.

Use the `<sub>` tag with the `alias` attribute to substitute a different word (or pronunciation) for selected text such as an acronym or abbreviation.

This uses the syntax:

```
<sub alias="new word">abbreviation</sub>
```

In the following example, the name "Mercury" is substituted for the element's chemical symbol to make the audio content clearer.

```
<speak>
  My favorite chemical element is <sub alias="Mercury">Hg</sub>, because it looks so
  shiny.
</speak>
```

Improving pronunciation by specifying parts of speech

<w>

This tag is supported by generative, long-form, neural, and standard TTS formats.

You can use the <w> tag to customize the pronunciation of words by specifying the word's part of speech or alternate meaning. This is done using the `role` attribute.

This tag uses the following syntax:

```
<w role="attribute">text</w>
```

The following values can be used for the `role` attribute:

To specify the part of speech:

- `amazon:VB`: interprets the word as a verb (present simple).
- `amazon:VBD`: interprets the word as past tense verb.
- `amazon:DT`: interprets the word as a determiner.
- `amazon:IN`: interprets the word as a preposition.
- `amazon:JJ`: interprets the word as an adjective.
- `amazon:NN`: interprets the word as a noun.

For example, depending on its part of speech, the US English pronunciation of the word "read" varies based on the tag:

```
<speak>
  The word <say-as interpret-as="characters">read</say-as> may be interpreted
  as either the present simple form <w role="amazon:VB">read</w>, or the past
  participle form <w role="amazon:VBD">read</w>.
```

```
</speak>
```

To specify a specific meaning:

- `amazon:DEFAULT`: uses the default sense of the word.
- `amazon:SENSE_1`: uses the non-default sense of the word when present. For example, the noun "bass" is pronounced differently depending on its meaning. The default meaning is the lowest part of the musical range. The alternate meaning is a species of freshwater fish, also called "bass" but pronounced differently. Using `<w role="amazon:SENSE_1">bass</w>` renders the non-default pronunciation (freshwater fish) for the audio text.

This difference in pronunciation and meaning can be heard if you synthesize the following:

```
<speak>
  Depending on your meaning, the word <say-as interpret-as="characters">bass</say-
as>
  may be interpreted as either a musical element: bass, or as its alternative
  meaning,
  a freshwater fish <w role="amazon:SENSE_1">bass</w>.
</speak>
```

Note

Some languages may have a different selection of supported parts of speech.

Adding the sound of breathing

<amazon:breath> and <amazon:auto-breaths>

This tag is supported only by the standard TTS format.

Natural-sounding speech includes both correctly spoken words and breathing sounds. By adding breathing sounds to synthesized speech, you can make it sound more natural. The `<amazon:breath>` and `<amazon:auto-breaths>` tags provide breaths. You have the following options:

- **Manual mode:** you set the location, length, and volume of a breath sound within the text
- **Automated mode:** Amazon Polly automatically inserts breathing sounds into the speech output

- Mixed mode: both you and Amazon Polly add breathing sounds

Manual Mode

In manual mode, you place the `<amazon:breath/>` tag in the input text where you want to locate a breath. You can customize the length and volume of breaths with the `duration` and `volume` attributes, respectively:

- `duration`: Controls the length of the breath. Valid values are: `default`, `x-short`, `short`, `medium`, `long`, `x-long`. The default value is `medium`.
- `volume`: Controls how loud breathing sounds. Valid values are: `default`, `x-soft`, `soft`, `medium`, `loud`, `x-loud`. The default value is `medium`.

Note

The exact length and volume of each attribute value is dependent on the specific Amazon Polly voice used.

To set a breath sound using the defaults, use `<amazon:breath/>` without attributes.

For example, to use attributes to set the duration and volume for a breath to medium, you would set the attributes as follows:

```
<speack>
    Sometimes you want to insert only <amazon:breath duration="medium" volume="x-
loud"/>a single breath.
</speack>
```

To use the defaults, you would just use the tag:

```
<speack>
    Sometimes you need <amazon:breath/>to insert one or more average breaths
    <amazon:breath/> so that the
    text sounds correct.
</speack>
```

You can add individual breathing sounds within a passage, as follows:

```
<speack>
  <amazon:breath duration="long" volume="x-loud"/> <prosody rate="120%"> <prosody
volume="loud">
  Wow! <amazon:breath duration="long" volume="loud"/> </prosody> That was quite
fast. <amazon:breath
  duration="medium" volume="x-loud"/> I almost beat my personal best time on this
track. </prosody>
</speack>
```

Automated Mode

In automated mode, you use the `<amazon:auto-breaths>` tag to tell Amazon Polly to automatically create breathing noises at appropriate intervals. You can set the frequency of the intervals, their volume, and their duration. Place the `</amazon:auto-breaths>` tag at the beginning of the text that you want to apply automated breathing to and then close the tag at the end.

Note

Unlike the manual mode tag, `<amazon:breath/>`, the `<amazon:auto-breaths>` tag requires a closing tag (`</amazon:auto-breaths>`).

You can use the following optional attributes with the `<amazon:auto-breaths>` tag:

- **volume:** Controls how loud the breathing sounds. Valid values are: `default`, `x-soft`, `soft`, `medium`, `loud`, `x-loud`. The default value is `medium`.
- **frequency:** Controls how often breathing sounds occur in the text. Valid values are: `default`, `x-low`, `low`, `medium`, `high`, `x-high`. The default value is `medium`.
- **duration:** Controls the length of the breath. Valid values are: `default`, `x-short`, `short`, `medium`, `long`, `x-long`. The default value is `medium`.

By default, the frequency of breathing sounds depends on the input text. However, breathing sounds often occur after commas and periods.

The following examples show how to use the `<amazon:auto-breaths>` tag. To decide which options to use for your content, copy the applicable examples to the Amazon Polly console and listen to the differences.

- Using automated mode without optional parameters.

```
<speaking>
    <amazon:auto-breaths>Amazon Polly is a service that turns text into lifelike
speech,
    allowing you to create applications that talk and build entirely new categories
of speech-
    enabled products. Amazon Polly is a text-to-speech service that uses advanced
deep learning
    technologies to synthesize speech that sounds like a human voice. With dozens of
lifelike
    voices across a variety of languages, you can select the ideal voice and build
speech-
    enabled applications that work in many different countries.</amazon:auto-
breaths>
</speaking>
```

- Using automated mode with volume control. The unspecified parameters (duration and frequency) are set to the default values (medium).

```
<speaking>
    <amazon:auto-breaths volume="x-soft">Amazon Polly is a service that turns text
into lifelike
    speech, allowing you to create applications that talk and build entirely new
categories of
    speech-enabled products. Amazon Polly is a text-to-speech service, that uses
advanced deep
    learning technologies to synthesize speech that sounds like a human voice. With
dozens of
    lifelike voices across a variety of languages, you can select the ideal voice
and build speech-
    enabled applications that work in many different countries.</amazon:auto-
breaths>
</speaking>
```

- Using automated mode with frequency control. The unspecified parameters (duration and volume) are set to the default values (medium).

```
<speaking>
    <amazon:auto-breaths frequency="x-low">Amazon Polly is a service that turns text
into lifelike
    speech, allowing you to create applications that talk and build entirely new
categories of
```

```

    speech-enabled products. Amazon Polly is a text-to-speech service, that uses
    advanced deep
    learning technologies to synthesize speech that sounds like a human voice. With
    dozens of
    lifelike voices across a variety of languages, you can select the ideal voice
    and build speech-
    enabled applications that work in many different countries.</amazon:auto-
    breaths>
</speak>

```

- Using automated mode with multiple parameters. For the unspecified Duration parameter, Amazon Polly uses the default value (medium).

```

<speak>
    <amazon:auto-breaths volume="x-loud" frequency="x-low">Amazon Polly is a service
    that turns
    text into lifelike speech, allowing you to create applications that talk and
    build entirely new
    categories of speech-enabled products. Amazon Polly is a text-to-speech service,
    that uses
    advanced deep learning technologies to synthesize speech that sounds like a
    human voice. With
    dozens of lifelike voices across a variety of languages, you can select the
    ideal voice and build
    speech-enabled applications that work in many different countries.</amazon:auto-
    breaths>
</speak>

```

Newscaster speaking style

<amazon:domain name="news">

The newscaster style is available only for the Matthew or Joanna voices, which are available only in American English (en-US), Lupe, in US Spanish (es-US) and Amy, in British English (en-GB). It is only supported when using Neural format.

To use the newscaster style, you use SSML tags and the following syntax::

```
<amazon:domain name="news">text</amazon:domain>
```

For example, you might use the newscaster style with the Amy voice as follows:

```

< speak >
< amazon:domain name="news">
From the Tuesday, April 16th, 1912 edition of The Guardian newspaper:

The maiden voyage of the White Star liner Titanic, the largest ship ever launched, has
ended in disaster.

The Titanic started her trip from Southampton for New York on Wednesday. Late on Sunday
night she struck
an iceberg off the Grand Banks of Newfoundland. By wireless telegraphy she sent out
signals of distress,
and several liners were near enough to catch and respond to the call.
</amazon:domain>
</speak>

```

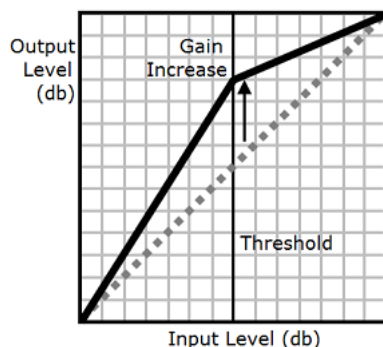
Adding dynamic range compression

```
<amazon:effect name="drc">
```

This tag is supported by long-form, neural, and standard TTS formats.

Depending on the text, language, and voice used in an audio file, the sounds range from soft to loud. Environmental sounds, such as the sound of a moving vehicle, can often mask the softer sounds, which makes the audio track difficult to hear clearly. To enhance the volume of certain sounds in your audio file, use the dynamic range compression (drc) tag.

The drc tag sets a midrange "loudness" threshold for your audio, and increases the volume (the gain) of the sounds around that threshold. It applies the greatest gain increase closest to the threshold, and the gain increase is lessened farther away from the threshold.



This makes the middle-range sounds easier to hear in a noisy environment, which makes the entire audio file clearer.

The `drc` tag is a Boolean parameter (it's either present or it isn't). It uses the syntax: `<amazon:effect name="drc">` and is closed with `</amazon:effect>`.

You can use the `drc` tag with any voice or language supported by Amazon Polly. You can apply it to an entire section of the recording, or for only a few words. For example:

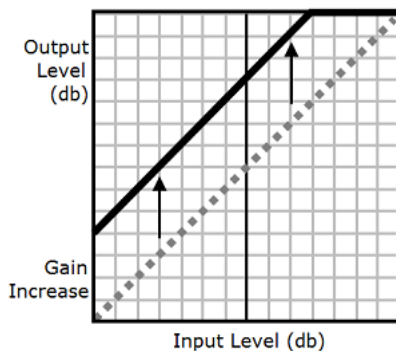
```
< speak >
  Some audio is difficult to hear in a moving vehicle, but <amazon:effect
name="drc"> this audio
  is less difficult to hear in a moving vehicle.</amazon:effect>
</ speak >
```

Note

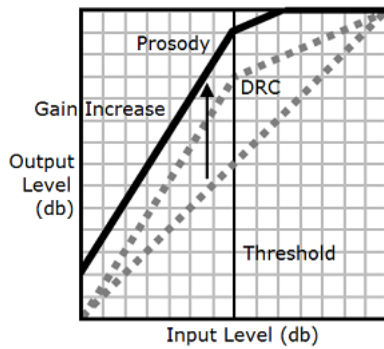
When you use "drc" in the `amazon:effect` syntax, it is case-sensitive.

Using `drc` with the `prosody volume` Tag

As the following graphic shows, the `prosody volume` tag evenly increases the volume of an entire audio file from the original level (dotted line) to an adjusted level (solid line). To further increase the volume of certain parts of the file, use the `drc` tag with the `prosody volume` tag. Combining tags doesn't affect the settings of the `prosody volume` tag.



When you use the `drc` and `prosody volume` tags together, Amazon Polly applies the `drc` tag first, increasing the middle-range sounds (those near the threshold). It then applies the `prosody volume` tag and further increases the volume of the entire audio track evenly.



To use the tags together, nest one inside the other. For example:

```
<speak>
  <prosody volume="loud">This text needs to be understandable and loud.
  <amazon:effect name="drc">
    This text also needs to be more understandable in a moving car.</amazon:effect></prosody>
</speak>
```

In this text, the `prosody volume` tag increases the volume of the entire passage to "loud." The `drc` tag enhances the volume of the middle-range values in the second sentence.

Note

When using the `drc` and `prosody volume` tags together, use standard XML practices for nesting tags.

Speaking softly

```
<amazon:effect phonation="soft">
```

This tag is currently supported only by the standard TTS format.

To specify that input text should be spoken in a softer-than-normal voice, use the `<amazon:effect phonation="soft">` tag.

This uses the syntax:

```
<amazon:effect phonation="soft">text</amazon:effect>
```

For example, you might use this tag with the Matthew voice as follows:

```
<speak>
  This is Matthew speaking in my normal voice. <amazon:effect phonation="soft">This
  is Matthew speaking in my softer voice.</amazon:effect>
</speak>
```

Controlling timbre

<amazon:effect vocal-tract-length>

This tag is currently supported only by the standard TTS format.

Timbre is the tonal quality of a voice that helps you tell the difference between voices, even when they have the same pitch and loudness. One of the most important physiological features that contributes to speech timbre is the length of the vocal tract. The vocal tract is a cavity of air that spans from the top of the vocal folds up to the edge of the lips.

To control the timbre of output speech in Amazon Polly, use the `vocal-tract-length` tag. This tag has the effect of changing the length of the speaker's vocal tract, which sounds like a change in the speaker's size. When you increase the `vocal-tract-length`, the speaker sounds physically bigger. When you decrease it, the speaker sounds smaller. You can use this tag with any of the voices in the Amazon Polly Text-to-Speech portfolio.

To change timbre, use the following values:

- `+n%` or `-n%`: Adjusts the vocal tract length by a relative percentage change in the current voice. For example, `+4%` or `-2%`. Valid values range from `+100%` to `-50%`. Values outside this range are clipped. For example, `+111%` sounds like `+100%` and `-60%` sounds like `-50%`.
- `n%`: Changes the vocal tract length to an absolute percentage of the tract length of the current voice. For example, `110%` or `75%`. An absolute value of `110%` is equivalent to a relative value of `+10%`. An absolute value of `100%` is the same as the default value for the current voice.

The following example shows how to change the vocal tract length to change timbre:

```
<speak>
  This is my original voice, without any modifications. <amazon:effect vocal-tract-
  length="+15%">
```

```
Now, imagine that I am much bigger. </amazon:effect> <amazon:effect vocal-tract-  
length="-15%">  
Or, perhaps you prefer my voice when I'm very small. </amazon:effect> You can also  
control the  
timbre of my voice by making minor adjustments. <amazon:effect vocal-tract-  
length="+10%">  
For example, by making me sound just a little bigger. </  
amazon:effect><amazon:effect  
vocal-tract-length="-10%"> Or, making me sound only somewhat smaller. </  
amazon:effect>  
</speak>
```

Combining Multiple Tags

You can combine the `vocal-tract-length` tag with any other SSML tag that is supported by Amazon Polly. Because timbre (vocal tract length) and pitch are closely connected, you might get the best results by using both the `vocal-tract-length` and the `<prosody pitch>` tags. To produce the most realistic voice, we recommend that you use different percentages of change for the two tags. Experiment with various combinations to get the results you want.

The following example shows how to combine tags.

```
<speak>  
The pitch and timbre of a person's voice are connected in human speech.  
<amazon:effect vocal-tract-length="-15%"> If you are going to reduce the vocal  
tract length,  
</amazon:effect><amazon:effect vocal-tract-length="-15%"> <prosody pitch="+20%">  
you  
might consider increasing the pitch, too. </prosody></amazon:effect>  
<amazon:effect vocal-tract-length="+15%"> If you choose to lengthen the vocal  
tract,  
</amazon:effect> <amazon:effect vocal-tract-length="+15%"> <prosody pitch="-10%">  
you might also want to lower the pitch. </prosody></amazon:effect>  
</speak>
```

Whispering

```
<amazon:effect name="whispered">
```

This tag is currently supported only by the standard TTS format.

This tag indicates that the input text should be spoken in a whispered voice rather than as normal speech. This can be used with any of the voices in the Amazon Polly Text-to-Speech portfolio.

This uses the following syntax:

```
<amazon:effect name="whispered">text</amazon:effect>
```

For example:

```
<speak>  
  <amazon:effect name="whispered">If you make any noise, </amazon:effect>  
  she said, <amazon:effect name="whispered">they will hear us.</amazon:effect>  
</speak>
```

In this case, the synthesized speech spoken by the character is whispered, but the phrase "she said" is spoken in the normal synthesized speech of the selected Amazon Polly voice.

You can enhance the "whispered" effect by slowing down the prosody rate by up to 10%, depending on the effect you want.

For example:

```
<speak>  
  When any voice is made to whisper, <amazon:effect name="whispered">  
    <prosody rate="-10%">the sound is slower and quieter than normal speech  
  </prosody></amazon:effect>  
</speak>
```

When generating speech marks for a whispered voice, the audio stream must also include the whispered voice to ensure that the speech marks match the audio stream.

Managing lexicons

Pronunciation lexicons enable you to customize the pronunciation of words. Amazon Polly provides API operations that you can use to store lexicons in an AWS region. Those lexicons are then specific to that particular region. You can use one or more of the lexicons from that region when synthesizing the text by using the `SynthesizeSpeech` operation. This applies the specified lexicon to the input text before the synthesis begins. For more information, see [SynthesizeSpeech](#).

Note

These lexicons must conform with the Pronunciation Lexicon Specification (PLS) W3C recommendation. For more information, see [Pronunciation Lexicon Specification \(PLS\) Version 1.0](#) on the W3C website.

The following are examples of ways to use lexicons with speech synthesis engines:

- Common words are sometimes stylized with numbers taking the place of letters, as with "g3t sm4rt" (get smart). Humans can read these words correctly. However, a Text-to-Speech (TTS) engine reads the text literally, pronouncing the name exactly as it is spelled. This is where you can leverage lexicons to customize the synthesized speech by using Amazon Polly. In this example, you can specify an alias (get smart) for the word "g3t sm4rt" in the lexicon.
- Your text might include an acronym, such as W3C. You can use a lexicon to define an alias for the word W3C so that it is read in the full, expanded form (World Wide Web Consortium).

Lexicons give you additional control over how Amazon Polly pronounces words uncommon to the selected language. For example, you can specify the pronunciation using a phonetic alphabet. For more information, see [Pronunciation Lexicon Specification \(PLS\) Version 1.0](#) on the W3C website.

Topics

- [Using multiple lexicons](#)
- [Uploading a lexicon](#)
- [Applying lexicons \(Synthesizing Speech\)](#)
- [Filtering the lexicon list on the console](#)
- [Downloading lexicons on the console](#)

- [Deleting a lexicon](#)

Using multiple lexicons

You can apply up to five lexicons to your text. If the same grapheme appears in more than one lexicon that you apply to your text, the order in which they are applied can make a difference in the resulting speech. For example, given the following text, "Hello, my name is Bob." and two lexemes in different lexicons that both use the grapheme Bob.

LexA

```
<lexeme>
  <grapheme>Bob</grapheme>
  <alias>Robert</alias>
</lexeme>
```

LexB

```
<lexeme>
  <grapheme>Bob</grapheme>
  <alias>Bobby</alias>
</lexeme>
```

If the lexicons are listed in the order LexA and then LexB, the synthesized speech will be "Hello, my name is Robert." If they are listed in the order LexB and then LexA, the synthesized speech is "Hello, my name is Bobby."

Example– Applying LexA Before LexB

```
aws polly synthesize-speech \
--lexicon-names LexA LexB \
--output-format mp3 \
--text 'Hello, my name is Bob' \
--voice-id Justin \
bobAB.mp3
```

Speech output: "Hello, my name is Robert."

Example– Applying LexB before LexA

```
aws polly synthesize-speech \  
--lexicon-names LexB LexA \  
--output-format mp3 \  
--text 'Hello, my name is Bob' \  
--voice-id Justin \  
bobBA.mp3
```

Speech output: "Hello, my name is Bobby."

For information about applying lexicons using the Amazon Polly console, see [Applying lexicons \(Synthesizing Speech\)](#).

Uploading a lexicon

The lexicons you use must conform to the Pronunciation Lexicon Specification (PLS) W3C recommendation. For more information, see [Pronunciation Lexicon Specification \(PLS\) Version 1.0](#) on the W3C website.

Console - Lexicons tab

To use a pronunciation lexicon, you must first upload it. There are two locations on the console from which you can upload a lexicon, the **Text-to-Speech** tab and the **Lexicons** tab.

The following processes describe how to add lexicons that you can use to customize how words and phrases uncommon to the chosen language are pronounced.

To add a lexicon from the Lexicons tab

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Lexicons** tab.
3. Choose **Upload lexicon**.
4. Provide a name for the lexicon and then use **Choose a lexicon file** to find the lexicon to upload. You can only upload PLS files with .pls or .xml extensions.
5. Choose **Upload lexicon**. If a lexicon by the same name (whether a .pls or .xml file) already exists, uploading the lexicon overwrites the existing lexicon.

Console - TTS tab

To add a lexicon from the text-to-Speech tab

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Text-to-Speech** tab.
3. Expand **Additional settings**, turn on **Customize pronunciation**, and then choose **Upload lexicon**.
4. Provide a name for the lexicon and then use **Choose a lexicon file** to find the lexicon to upload. You can only use PLS files with .pls or .xml extensions.
5. Choose **Upload lexicon**. If a lexicon with the same name (whether a .pls or .xml file) already exists, uploading the lexicon overwrites the existing lexicon.

AWS CLI - one lexeme

With Amazon Polly, you can use [PutLexicon](#) to store pronunciation lexicons in a specific AWS Region for your account. Then, you can specify one or more of these stored lexicons in your [SynthesizeSpeech](#) request that you want to apply before the service starts synthesizing the text. For more information, see [Managing lexicons](#).

Consider the following W3C PLS-compliant lexicon.

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
  alphabet="ipa"
  xml:lang="en-US">
  <lexeme>
    <grapheme>W3C</grapheme>
    <alias>World Wide Web Consortium</alias>
  </lexeme>
</lexicon>
```

Note the following:

- The two attributes specified in the <lexicon> element:

- The `xml:lang` attribute specifies the language code, `en-US`, to which the lexicon applies. Amazon Polly can use this example lexicon if the voice you specify in the `SynthesizeSpeech` call has the same language code (`en-US`).

 Note

You can use the `DescribeVoices` operation to find the language code associated with a voice.

- The `alphabet` attribute specifies IPA, which means that the International Phonetic Alphabet (IPA) alphabet is used for pronunciations. IPA is one of the alphabets for writing pronunciations. Amazon Polly also supports the Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA).
- The `<lexeme>` element describes the mapping between `<grapheme>` (that is, a textual representation of the word) and `<alias>`.

To test this lexicon, do the following:

1. Save the lexicon as `example.pls`.
2. Run the `put-lexicon` AWS CLI command to store the lexicon (with the name `w3c`), in the `us-east-2` region.

```
aws polly put-lexicon \  
--name w3c \  
--content file://example.pls
```

3. Run the `synthesize-speech` command to synthesize sample text to an audio stream (`speech.mp3`), and specify the optional `lexicon-name` parameter.

```
aws polly synthesize-speech \  
--text 'W3C is a Consortium' \  
--voice-id Joanna \  
--output-format mp3 \  
--lexicon-names="w3c" \  

```

```
speech.mp3
```

4. Play the resulting `speech.mp3`, and notice that the word W3C in the text is replaced by World Wide Web Consortium.

The preceding example lexicon uses an alias. The IPA alphabet mentioned in the lexicon is not used. The following lexicon specifies a phonetic pronunciation using the `<phoneme>` element with the IPA alphabet.

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
  alphabet="ipa"
  xml:lang="en-US">
  <lexeme>
    <grapheme>pecan</grapheme>
    <phoneme>p##k##n</phoneme>
  </lexeme>
</lexicon>
```

Follow the same steps to test this lexicon. Make sure you specify input text that has the word "pecan" (for example, "Pecan pie is delicious").

See the following resources for additional code samples for the PutLexicon API operation:

- Java Sample: [PutLexicon](#)
- Python (Boto3) Sample: [PutLexicon](#)

AWS CLI - multiple lexemes

With Amazon Polly, you can use [PutLexicon](#) to store pronunciation lexicons in a specific AWS Region for your account. Then, you can specify one or more of these stored lexicons in your [SynthesizeSpeech](#) request that you want to apply before the service starts synthesizing the text. For more information, see [Managing lexicons](#).

In this example, the lexeme that you specify in the lexicon applies exclusively to the input text for the synthesis. Consider the following lexicon:

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
  alphabet="ipa" xml:lang="en-US">

  <lexeme>
    <grapheme>W3C</grapheme>
    <alias>World Wide Web Consortium</alias>
  </lexeme>
  <lexeme>
    <grapheme>W3C</grapheme>
    <alias>WWW Consortium</alias>
  </lexeme>
  <lexeme>
    <grapheme>Consortium</grapheme>
    <alias>Community</alias>
  </lexeme>
</lexicon>
```

The lexicon specifies three lexemes, two of which define an alias for the grapheme W3C as follows:

- The first `<lexeme>` element defines an alias (World Wide Web Consortium).
- The second `<lexeme>` defines an alternative alias (WWW Consortium).

Amazon Polly uses the first replacement for any given grapheme in a lexicon.

The third `<lexeme>` defines a replacement (Community) for the word Consortium.

First, let's test this lexicon. Suppose you want to synthesize the following sample text to an audio file (`speech.mp3`), and you specify the lexicon in a call to `SynthesizeSpeech`.

The W3C is a Consortium

`SynthesizeSpeech` first applies the lexicon as follows:

- As per the first lexeme, the word W3C is revised as World Wide Web Consortium. The revised text appears as follows:

```
The World Wide Web Consortium is a Consortium
```

- The alias defined in the third lexeme applies only to the word Consortium that was part of the original text, resulting in the following text:

```
The World Wide Web Consortium is a Community.
```

You can test this using the AWS CLI as follows:

1. Save the lexicon as `example.pls`.
2. Run the `put-lexicon` command to store the lexicon with name `w3c` in the `us-east-2` region.

```
aws polly put-lexicon \  
--name w3c \  
--content file://example.pls
```

3. Run the `list-lexicons` command to verify that the `w3c` lexicon is in the list of lexicons returned.

```
aws polly list-lexicons
```

4. Run the `synthesize-speech` command to synthesize sample text to an audio file (`speech.mp3`), and specify the optional `lexicon-name` parameter.

```
aws polly synthesize-speech \  
--text 'W3C is a Consortium' \  
--voice-id Joanna \  
--output-format mp3 \  
--lexicon-names="w3c" \  
speech.mp3
```

5. Play the resulting `speech.mp3` file to verify that the synthesized speech reflects the text changes.

See the following resources for additional code samples for the `PutLexicon` API operation:

- Java Sample: [PutLexicon](#)

- Python (Boto3) Sample: [PutLexicon](#)

Applying lexicons (Synthesizing Speech)

The lexicons you use must conform to the Pronunciation Lexicon Specification (PLS) W3C recommendation. For more information, see [Pronunciation Lexicon Specification \(PLS\) Version 1.0](#) on the W3C website.

Console

The following procedure demonstrates how to apply a lexicon to your input text by applying the `W3c.pls` lexicon to substitute "World Wide Web Consortium" for "W3C". If you apply multiple lexicons to your text they are applied in a top-down order with the first match taking precedence over later matches. A lexicon is applied to the text only if the language specified in the lexicon is the same as the language chosen.

You can apply a lexicon to plain text or SSML input.

Example– Applying the W3C.pls Lexicon

To create the lexicon you'll need for this exercise, see [Uploading a lexicon](#). Use a plain text editor to create the `W3C.pls` lexicon shown at the top of the topic. Remember where you save this file.

To apply the W3C.pls lexicon to your input

In this example we introduce a lexicon to substitute "World Wide Web Consortium" for "W3C". Compare the results of this exercise with that of [Using SSML on the console](#) for both US English and another language.

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Do one of the following:
 - Turn off **SSML** and then type or paste this text into the text input box.

```
He was caught up in the game.  
In the middle of the 10/3/2014 W3C meeting  
he shouted, "Score!" quite loudly.
```

- Turn on **SSML** and then type or paste this text into the text input box.

```
< speak>He wasn't paying attention.< break time="1s"/>
In the middle of the 10/3/2014 W3C meeting
he shouted, "Score!" quite loudly.</ speak>
```

3. From the **Language** list, choose **English, US**, then choose the voice you want to use for this text.
4. Expand **Additional settings** and turn on **Customize pronunciation**.
5. From the list of lexicons, choose W3C (English, US).

If the W3C (English, US) lexicon is not listed, choose **Upload lexicon** and upload it, then choose it from the list. To create this lexicon, see [Uploading a lexicon](#).

6. To listen to the speech immediately, choose **Listen**.
7. To save the speech to a file,
 - a. Choose **Download**.
 - b. To change to a different file format, turn on **Speech file format settings**, choose the file format you want, and then choose **Download**.

Repeat the previous steps, but choose a different language and notice the difference in the output.

AWS CLI

In a call to `SynthesizeSpeech`, you can specify multiple lexicons. In this case, the first lexicon specified (in order from left to right) overrides any preceding lexicons.

Consider the following two lexicons. Note that each lexicon describes different aliases for the same grapheme W3C.

- Lexicon 1: `w3c.pls`

```
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
```

```

    alphabet="ipa" xml:lang="en-US">
<lexeme>
  <grapheme>W3C</grapheme>
  <alias>World Wide Web Consortium</alias>
</lexeme>
</lexicon>

```

- Lexicon 2: w3cAlternate.pls

```

<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
  xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
  alphabet="ipa" xml:lang="en-US">

  <lexeme>
    <grapheme>W3C</grapheme>
    <alias>WWW Consortium</alias>
  </lexeme>
</lexicon>

```

Suppose you store these lexicons as w3c and w3cAlternate respectively. If you specify lexicons in order (w3c followed by w3cAlternate) in a SynthesizeSpeech call, the alias for W3C defined in the first lexicon has precedence over the second. To test the lexicons, do the following:

1. Save the lexicons locally in files called w3c.pls and w3cAlternate.pls.
2. Upload these lexicons using the put-lexicon AWS CLI command.
 - Upload the w3c.pls lexicon and store it as w3c.

```

aws polly put-lexicon \
--name w3c \
--content file://w3c.pls

```

- Upload the w3cAlternate.pls lexicon on the service as w3cAlternate.

```

aws polly put-lexicon \

```



```
--name w3cAlternate \  
--content file://w3cAlternate.pls
```

3. Run the `synthesize-speech` command to synthesize sample text to an audio stream (`speech.mp3`), and specify both lexicons using the `lexicon-name` parameter.

```
aws polly synthesize-speech \  
--text 'PLS is a W3C recommendation' \  
--voice-id Joanna \  
--output-format mp3 \  
--lexicon-names '["w3c","w3cAlternative"]' \  
speech.mp3
```

4. Test the resulting `speech.mp3`. It should read as follows:

```
PLS is a World Wide Web Consortium recommendation
```

Filtering the lexicon list on the console

The following procedure describes how to filter the lexicons list so that only lexicons of a chosen language are displayed.

Console

To filter the lexicons listed by language

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Lexicons** tab.
3. Choose **Any language**.
4. From the list of languages, choose the language you want to filter on.

The list displays only the lexicons for the chosen language.

AWS CLI

Amazon Polly provides the [ListLexicons](#) API operation that you can use to get the list of pronunciation lexicons in your account in a specific AWS Region. The following AWS CLI call lists the lexicons in your account in the `us-east-2` region.

```
aws polly list-lexicons
```

The following is an example response, showing two lexicons named `w3c` and `tomato`. For each lexicon, the response returns metadata such as the language code to which the lexicon applies, the number of lexemes defined in the lexicon, the size in bytes, and so on. The language code describes a language and locale to which the lexemes defined in the lexicon apply.

```
{
  "Lexicons": [
    {
      "Attributes": {
        "LanguageCode": "en-US",
        "LastModified": 1474222543.989,
        "Alphabet": "ipa",
        "LexemesCount": 1,
        "LexiconArn": "arn:aws:polly:aws-region:account-id:lexicon/w3c",
        "Size": 495
      },
      "Name": "w3c"
    },
    {
      "Attributes": {
        "LanguageCode": "en-US",
        "LastModified": 1473099290.858,
        "Alphabet": "ipa",
        "LexemesCount": 1,
        "LexiconArn": "arn:aws:polly:aws-region:account-id:lexicon/tomato",
        "Size": 645
      },
      "Name": "tomato"
    }
  ]
}
```

The following resources contain additional information for the `ListLexicons` operation:

- Java Sample: [ListLexicons](#)
- Python (Boto3) Sample: [ListLexicon](#)

Downloading lexicons on the console

The following process describes how to download one or more lexicons. You can add, remove, or modify lexicon entries in the file and then upload it again to keep your lexicon up-to-date.

Console

To download one or more lexicons

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Lexicons** tab.
3. Choose the lexicon or lexicons you want to download.
 - a. To download a single lexicon, choose its name from the list.
 - b. To download multiple lexicons as a single compressed archive file, select the check box next to each entry in the list that you want to download.
4. Choose **Download**.
5. Open the folder where you want to download the lexicon.
6. Choose **Save**.

AWS CLI

Amazon Polly provides the [GetLexicon](#) API operation to retrieve the content of a pronunciation lexicon you stored in your account in a specific region.

The following `get-lexicon` AWS CLI command retrieves the content of the example lexicon.

```
aws polly get-lexicon \  
--name example
```

If you don't already have a lexicon stored in your account, you can use the `PutLexicon` operation to store one. For more information, see [Uploading a lexicon](#).

The following is a sample response. In addition to the lexicon content, the response returns the metadata, such as the language code to which the lexicon applies, number of lexemes defined in the lexicon, the Amazon Resource Name (ARN) of the resource, and the size of the lexicon in bytes. The `LastModified` value is a Unix timestamp.

```
{
  "Lexicon": {
    "Content": "lexicon content in plain text PLS format",
    "Name": "example"
  },
  "LexiconAttributes": {
    "LanguageCode": "en-US",
    "LastModified": 1474222543.989,
    "Alphabet": "ipa",
    "LexemesCount": 1,
    "LexiconArn": "arn:aws:polly:us-east-2:account-id:lexicon/example",
    "Size": 495
  }
}
```

The following resources contain additional code samples for the GetLexicon operation:

- Java Sample: [GetLexicon](#)
- Python (Boto3) Sample: [GetLexicon](#)

Deleting a lexicon

The following process describes how to delete a lexicon. After deleting the lexicon, you must add it back before you can use it again. You can delete one or more lexicons at the same time by selecting the check boxes next to individual lexicons.

Console

To delete a lexicon

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Lexicons** tab.
3. Choose one or more lexicons that you want to delete from the list.
4. Choose **Delete**.
5. Enter confirmation text and then choose **Delete** to remove the lexicon from the Region or **Cancel** to keep it.

AWS CLI

Amazon Polly provides the [DeleteLexicon](#) API operation to delete a pronunciation lexicon from a specific AWS Region in your account. The following AWS CLI deletes the specified lexicon.

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly delete-lexicon \  
--name example
```

The following resources contain additional information for the DeleteLexicon operation:

- Java Sample: [DeleteLexicon](#)
- Python (Boto3) Sample: [DeleteLexicon](#)

Long audio files

To create TTS files for large passages of text, use Amazon Polly's *asynchronous synthesis* functionality. This uses the three `SpeechSynthesisTask` APIs:

- `StartSpeechSynthesisTask`: starts a new synthesis task.
- `GetSpeechSynthesisTask`: returns details about a previously submitted synthesis task.
- `ListSpeechSynthesisTasks`: lists all submitted synthesis tasks.

The `SynthesizeSpeech` operation produces audio in near-real time, with relatively little latency in most cases. To do this, the operation can only synthesize 3000 characters.

Amazon Polly's Asynchronous Synthesis feature overcomes the challenge of processing a larger text document by changing the way the document is both synthesized and returned. When a synthesis request is made by submitting input text using the `StartSpeechSynthesisTask`, Amazon Polly queues the requests, and then asynchronously processes them in the background as soon as the system resources are available. Amazon Polly then uploads the resulting speech or speech marks stream directly to your (required) Amazon Simple Storage Service (Amazon S3) bucket, and notifies you about the completed file's availability through your (optional) SNS topic.

In this way, all of the functionality except near-real time processing is available for texts of up to 100,000 billable characters (or 200,000 total characters) in length.

To synthesize a document using this method, you must have an Amazon S3 bucket that is writable to which the audio file can be saved. You can be notified when the synthesized audio is ready by providing an optional SNS Topic identifier. When the synthesis task is complete, Amazon Polly will publish a message on that topic. This message may also contain useful error information in cases where the synthesis task didn't succeed. To do this, make sure that the user creating the synthesis task can also publish to the SNS Topic. See the [Amazon SNS documentation](#) for more information on how to create and subscribe to an SNS Topic.

Encryption

You can store the output file in an encrypted form in your S3 bucket if desired. To do this, you enable [Amazon S3 bucket encryption](#), which use one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256).

Topics

- [Setting up the IAM policy for asynchronous synthesis](#)
- [Creating long audio files](#)

Setting up the IAM policy for asynchronous synthesis

In order to use the asynchronous synthesis functionality, you will need an IAM policy that allows the following:

- use of new Amazon Polly operations
- writing to the output S3 bucket
- publishing to the status SNS topic [optional]

The following policy grants only the necessary permissions required for asynchronous synthesis and can be attached to the IAM user.

Creating long audio files

You can use the Amazon Polly console to create long speeches using asynchronous synthesis with the same functionality as you can use with the AWS CLI. This is done using the **Text-to-Speech** tab much like any other synthesis.

Console

The other asynchronous synthesis functionality is also available via the console. The **S3 synthesis tasks** tab reflects the `ListSpeechSynthesisTasks` functionality, displaying all tasks saved to the S3 bucket and enabling you to filter them if you want. Clicking on a specific single task shows its details, reflecting `GetSpeechSynthesisTask` functionality.

To synthesize a large text using the Amazon Polly console

1. Sign in to the AWS Management Console and open the Amazon Polly console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/polly/>.
2. Choose the **Text-to-Speech** tab. Select **Long Form** as the engine if appropriate.
3. With **SSML** on or off, type or paste your text into the input box.
4. Choose the language, region, and voice for your text.
5. Choose **Save to S3**.

Note

Both the **Download** and **Listen** options are greyed out if the text length is above the 3,000 character limit for the real-time SynthesizeSpeech operation.

6. The console opens a form so that you can choose where to store the output file.
 - a. Fill in the name of the destination Amazon S3 bucket.
 - b. Optionally, fill in the prefix key of the output.

Note

The output S3 bucket must be writable.

- c. If you want to be notified when the synthesis task is complete, provide an optional SNS topic identifier.

Note

The SNS must be open for publication by the current console user to use this option. For more information, see [Amazon Simple Notification Service \(SNS\)](#)

- d. Choose **Save to S3**.

To retrieve information on your speech synthesis tasks

1. In the console, choose the **S3 Synthesis Tasks** tab.
2. The tasks are displayed in date order. To filter the tasks, by status, choose **All statuses** and then choose the status to use.
3. To view the details of a specific task, choose the linked **Task ID**.

AWS CLI

Amazon Polly *asynchronous synthesis* functionality uses three SpeechSynthesisTask APIs to work with large amounts of text:

- **StartSpeechSynthesisTask**: starts a new synthesis task.

- `GetSpeechSynthesisTask`: returns details about a previously submitted synthesis task.
- `ListSpeechSynthesisTasks`: lists all submitted synthesis tasks.

Synthesizing large amounts of text (`StartSpeechSynthesisTask`)

When you want to create an audio file larger than one that you can create with the real-time `SynthesizeSpeech`, use the `StartSpeechSynthesisTask` operation. In addition to the arguments needed for the `SynthesizeSpeech` operation, `StartSpeechSynthesisTask` also requires the name of an Amazon S3 bucket. Two other optional arguments are also available: a key prefix for the output file and the ARN for an SNS Topic if you want to receive status notification about the task.

- `OutputS3BucketName`: The name of the Amazon S3 bucket where the synthesis should be uploaded. This bucket should be in the same region as the Amazon Polly service. Additionally, the IAM user being used to make the call should have access to the bucket. [Required]
- `OutputS3KeyPrefix`: Key prefix for the output file. Use this parameter if you want to save the output speech file in a custom directory-like key in your bucket. [Optional]
- `SnsTopicArn`: The SNS topic ARN to use if you want to receive notifications about status of the task. This SNS topic should be in the same region as the Amazon Polly service. Additionally, the IAM user being used to make the call should have access to the topic. [Optional]

For example, the following example can be used to run the `start-speech-synthesis-task` AWS CLI command in the US East (Ohio) region:

The following AWS CLI 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 (^) and use full quotation marks (") around the input text with single quotes (') for interior tags.

```
aws polly start-speech-synthesis-task \  
  --region us-east-2 \  
  --endpoint-url "https://polly.us-east-2.amazonaws.com/" \  
  --output-format mp3 \  
  --output-s3-bucket-name your-bucket-name \  
  --output-s3-key-prefix optional/prefix/path/file \  
  --voice-id Joanna \  
  --text file://text_file.txt
```

This will result in a response that looks similar to this:

```
"SynthesisTask":
{
  "OutputFormat": "mp3",
  "OutputUri": "https://s3.us-east-2.amazonaws.com/your-bucket-name/optional/
prefix/path/file.<task_id>.mp3",
  "TextType": "text",
  "CreationTime": [...],
  "RequestCharacters": [...],
  "TaskStatus": "scheduled",
  "TaskId": [task_id],
  "VoiceId": "Joanna"
}
```

The `start-speech-synthesis-task` operation returns several new fields:

- **OutputUri**: the location of your output speech file.
- **TaskId**: a unique identifier for the speech synthesis task generated by Amazon Polly.
- **CreationTime**: a timestamp for when the task was initially submitted.
- **RequestCharacters**: the number of billable characters in the task.
- **TaskStatus**: provides information on the status of the submitted task.

When your task is submitted, the initial status will show `scheduled`. When Amazon Polly starts processing the task, the status will change to `inProgress` and later, to `completed` or `failed`. If the task fails, an error message will be returned when calling either the `GetSpeechSynthesisTask` or `ListSpeechSynthesisTasks` operation.

When the task is completed, the speech file is available at the location specified in **OutputUri**.

Retrieving information on your speech synthesis task

You can get information on a task, such as errors, status, and so on, using the `GetSpeechSynthesisTask` operation. To do this, you will need the `task-id` returned by the `StartSpeechSynthesisTask`.

For example, the following example can be used to run the `get-speech-synthesis-task` AWS CLI command:

```
aws polly get-speech-synthesis-task \
```

```
--region us-east-2 \  
--endpoint-url "https:// polly.us-east-2.amazonaws.com/" \  
--task-id task identifier
```

You can also list all speech synthesis tasks that you've run in the current region using the `ListSpeechSynthesisTasks` operation.

For example, the following example can be used to run the `list-speech-synthesis-tasks` AWS CLI command:

```
aws polly list-speech-synthesis-tasks \  
--region us-east-2 \  
--endpoint-url "https:// polly.us-east-2.amazonaws.com/"
```

Quotas in Amazon Polly

Amazon Polly applies quotas to customer traffic by rejecting excessive requests. The default quota for the `SynthesizeSpeech` request with standard voices is 80 transactions per second (tps), in a single region, for a single AWS account. If limits did not increase, and if you generated 100 `SynthesizeSpeech` requests per second using a standard voice, 80 requests per second would succeed, and 20 requests per second would be throttled by Amazon Polly. These requests would return a response with HTTP status 400, and a response header indicating `ThrottlingException`. Amazon Polly also throttles traffic to all operations based on the request rate.

Speech synthesis limit examples

- **Synthesize the first 24 letters of the English alphabet one letter at a time.** If the synthesis of each letter took less than 50 milliseconds, with an operation limit of eight tps, synthesizing 24 letters would take at least three seconds. During that time, you could synthesize up to eight letters per second. Any further requests would be throttled. As the requests last a short time, they would be synthesized serially without overlap.
- **Synthesize 16 paragraphs of text.** If each paragraph was synthesized and fully received on the client side in two seconds or less, with an operation limit of eight concurrent requests, it would take at least four seconds to synthesize all 16 articles. In the first second, you could start up to eight requests. During concurrent requests, any attempt to start a new synthesis would be throttled due to the concurrency limit. You could synthesize the remaining eight paragraphs after the first two seconds, after the first batch of requests finishes.

Keep the following limits in mind when using Amazon Polly.

Topics

- [Supported regions](#)
- [Quotas and throttle rates](#)
- [Pronunciation lexicons](#)
- [SynthesizeSpeech API operations](#)
- [SpeechSynthesisTask API operations](#)
- [Speech Synthesis Markup Language \(SSML\)](#)

Supported regions

For a list of AWS Regions where Amazon Polly is available, see [Amazon Polly Endpoints and Quotas](#) in the *Amazon Web Services General Reference*.

- For Regions that support generative voices, see [Generative voices](#).
- For Regions that support long-form voices, see [Long-form voices](#).
- For Regions that support neural voices, see [the section called “Feature and region compatibility”](#) for neural TTS.

Quotas and throttle rates

The following table defines throttle rates per Amazon Polly operation. You can use the AWS Management Console to request quota increases for the adjustable quotas when needed.

Operation	Limit
Lexicon	
DeleteLexicon	Any 5 transactions per second (tps) from these operations combined. Maximum allowed burst of 10 tps.
PutLexicon	
GetLexicon	
ListLexicons	
Speech	
DescribeVoices	80 tps with a burst limit of 100 tps
SynthesizeSpeech	Generative voice: 8 tps Long-form voice: 8 tps with a burst limit of 10 tps Neural voice: 8 tps with a burst limit of 10 tps Standard voice: 80 tps with a burst limit of 100 tps

Operation	Limit
<code>StartSpeechSynthesisTask</code>	Generative voice: 1 tps Long-form voice: 1 tps Neural voice: 10 tps Standard voice: 10 tps with a burst limit of 12 tps
<code>StartSpeechSynthesisStream</code>	Generative voice: 8 tps
<code>GetSynthesizeSpeechTask</code> and <code>ListSynthesizeSpeechTask</code>	Maximum allowed 10 tps combined

Concurrent requests

For **generative voice**, Amazon Polly supports up to 26 concurrent requests. For **long-form voice**, Amazon Polly supports up to 26 concurrent requests. For **neural voice**, Amazon Polly supports 8 tps with a burst limit of 10 tps, for up to 18 concurrent requests. Amazon Polly also supports limits for concurrent requests. For **standard voice**, Amazon Polly supports 80 tps for up to 80 concurrent requests.

For **StartSpeechSynthesisStream**, Amazon Polly supports up to 8 concurrent requests.

Best practices to mitigate throttling

- **Retry throttles with backoff and jitter** so you can spread the load over a short period of time, and handle unexpected peaks in usage without compromising availability. AWS Code Sample Catalog is already configured to do this by default in many programming languages. Visit [feature retry behavior](#) to see the details.
- **Use [Amazon Polly metrics](#)**. Amazon Polly automatically publishes to CloudWatch to analyze your current usage and forecast usage growth.

Note

Before requesting a quota increase (where applicable), calculate your tps needs following the guidelines on this page. Amazon Polly secures only the required computational resources according to customer demand in order to keep your costs low.

Pronunciation lexicons

- You can store up to 100 lexicons per account.
- Lexicon names can be an alphanumeric string up to 20 characters long.
- Each lexicon can be up to 40,000 characters in size. (Note that the size of the lexicon affects the latency of the SynthesizeSpeech operation.)
- You can specify up to 100 characters for each <phoneme> or <alias> replacement in a lexicon.

For information about using lexicons, see [Managing lexicons](#).

SynthesizeSpeech API operations

When estimating the usage of SynthesizeSpeech, keep in mind that the audio produced by Amazon Polly, especially for interactive applications, usually takes at least several seconds to be played. This reduces the rate of requests to SynthesizeSpeech, even for a large number of concurrent consumers. Additionally, Amazon Polly throttles SynthesizeSpeech requests by the number of concurrent requests that it synthesizes. There is no separate setting for concurrent requests. The concurrent requests limit has always the same value as the number of tps allowed and scales with it.

Short story example application. You can use Amazon Polly to build an application that plays a series of short stories. With this kind of app, the first story would start playing, and then the next, and so on, until a user quit the application. Each story would take around 0.5 seconds to synthesize and 10 seconds to play. In this scenario, you could expect one call to SynthesizeSpeech for every 10 seconds that the customer spent using the application. This would translate to one call per second for every 10 customers who were concurrently using the application. If you had 1000 customers concurrently using the application, you could expect an average call rate to SynthesizeSpeech of only 100 transactions per second.

Note the following limits related to using the SynthesizeSpeech API operation:

- The size of the input text can be up to 3000 billed characters (6000 total characters). SSML tags are not counted as billed characters.
- You can specify up to five lexicons to apply to the input text.
- The output audio stream (synthesis) is limited to 10 minutes. After this is reached, any remaining speech is cut off.

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

 Note

Some limitations of the SynthesizeSpeech API operation can be bypassed using the StartSynthesizeSpeechTask API operation. For more information, see [Long audio files](#).

SpeechSynthesisTask API operations

Note the following limit relating to using the StartSpeechSynthesisTask, GetSpeechSynthesisTask, and ListSpeechSynthesisTasks API operations:

- The size of the input text can be up to 100,000 billed characters (200,000 total characters). SSML tags are not counted as billed characters.
- You can specify up to five lexicons to apply to the input text.

Speech Synthesis Markup Language (SSML)

Note the following limits related to using SSML:

- The <audio>, <lexicon>, <lookup>, and <voice> tags are not supported.
- <break> elements can specify a maximum duration of 10 seconds each.
- The <prosody> tag doesn't support values for the rate attribute lower than -80%.

For more information, see [Generating speech from SSML documents](#).

Sample code and applications for Amazon Polly

This section provides code samples and example applications that you can use to explore Amazon Polly.

The **Sample Code** topic contains snippets of code organized by programming language and separated into examples for different Amazon Polly functionality. The **Example Application** topic contains applications organized by programming language that can be used independently to explore Amazon Polly.

Before you start using these examples, we recommend that you first read [How Amazon Polly works](#) and follow the steps described in [Getting started with Amazon Polly](#).

Topics

- [Java samples](#)
- [Python samples](#)
- [Java example](#)
- [Python example \(HTML5 Client and Python Server\)](#)
- [iOS example](#)
- [Android example](#)

Java samples

The following code samples show how to use Java-based applications to accomplish various tasks with Amazon Polly. These samples are not full examples, but can be included in larger Java applications that use the [AWS SDK for Java](#).

Code Snippets

- [DeleteLexicon](#)
- [DescribeVoices](#)
- [GetLexicon](#)
- [ListLexicons](#)
- [PutLexicon](#)
- [StartSpeechSynthesisTask](#)

- [Speech Marks](#)
- [SynthesizeSpeech](#)

DeleteLexicon

The following Java code sample show how to use Java-based applications to delete a specific lexicon stored in an AWS Region. A lexicon which has been deleted is not available for speech synthesis, nor can it be retrieved using either the `GetLexicon` or `ListLexicon` APIs.

For more information on this operation, see the reference for the [DeleteLexicon](#) API.

SDK v2

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */

package com.example.polly;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.DeleteLexiconRequest;
import software.amazon.awssdk.services.polly.model.DeleteLexiconResponse;
import software.amazon.awssdk.services.polly.model.PollyException ;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class DeleteLexiconSample {

    public static void main(String args[]) {

        PollyClient polly = PollyClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create())
```

```

        .build();

        deleteLexicon(polly) ;
        polly.close();
    }

    private static String LEXICON_NAME = "SampleLexicon";
    public static void deleteLexicon(PollyClient client) {

        try {
            DeleteLexiconRequest deleteLexiconRequest = DeleteLexiconRequest.builder()
                .name(LEXICON_NAME).build();

            DeleteLexiconResponse deleteLexiconResult =
client.deleteLexicon(deleteLexiconRequest);

        } catch (PollyException e) {
            System.err.println("Exception caught: " + e);
            System.exit(1);
        }
    }
}

```

SDK v1

```

package com.amazonaws.polly.samples;

import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.AmazonPollyClientBuilder;
import com.amazonaws.services.polly.model.DeleteLexiconRequest;

public class DeleteLexiconSample {
    private String LEXICON_NAME = "SampleLexicon";

    AmazonPolly client = AmazonPollyClientBuilder.defaultClient();

    public void deleteLexicon() {
        DeleteLexiconRequest deleteLexiconRequest = new
DeleteLexiconRequest().withName(LEXICON_NAME);

        try {
            client.deleteLexicon(deleteLexiconRequest);

```

```
        } catch (Exception e) {  
            System.err.println("Exception caught: " + e);  
        }  
    }  
}
```

DescribeVoices

The following Java code sample shows how to use Java-based applications to produce a list of the voices that are available for use when requesting speech synthesis. You can optionally specify a language code to filter the available voices. For example, if you specify en-US, the operation returns a list of all available US English voices.

For more information on this operation, see the reference for the [DescribeVoices](#) API.

```
package com.amazonaws.polly.samples;  
  
import com.amazonaws.services.polly.AmazonPolly;  
import com.amazonaws.services.polly.AmazonPollyClientBuilder;  
import com.amazonaws.services.polly.model.DescribeVoicesRequest;  
import com.amazonaws.services.polly.model.DescribeVoicesResult;  
  
public class DescribeVoicesSample {  
    AmazonPolly client = AmazonPollyClientBuilder.defaultClient();  
  
    public void describeVoices() {  
        DescribeVoicesRequest allVoicesRequest = new DescribeVoicesRequest();  
        DescribeVoicesRequest enUsVoicesRequest = new  
DescribeVoicesRequest().withLanguageCode("en-US");  
  
        try {  
            String nextToken;  
            do {  
                DescribeVoicesResult allVoicesResult =  
client.describeVoices(allVoicesRequest);  
                nextToken = allVoicesResult.getNextToken();  
                allVoicesRequest.setNextToken(nextToken);  
  
                System.out.println("All voices: " + allVoicesResult.getVoices());  
            } while (nextToken != null);  
  
            do {
```

```

        DescribeVoicesResult enUsVoicesResult =
client.describeVoices(enUsVoicesRequest);
        nextToken = enUsVoicesResult.getNextToken();
        enUsVoicesRequest.setNextToken(nextToken);

        System.out.println("en-US voices: " + enUsVoicesResult.getVoices());
    } while (nextToken != null);
} catch (Exception e) {
    System.err.println("Exception caught: " + e);
}
}
}

```

GetLexicon

The following Java code sample show how to use Java-based applications to produce the content of a specific pronunciation lexicon stored in a AWS Region.

For more information on this operation, see the reference for the [GetLexicon](#) API.

SDK v2

```

/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */

package com.example.polly;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.GetLexiconRequest;
import software.amazon.awssdk.services.polly.model.GetLexiconResponse;
import software.amazon.awssdk.services.polly.model.PollyException ;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

```

```

*/
public class GetLexiconSample {

    public static void main(String args[]) {

        PollyClient polly = PollyClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        getLexicon(polly) ;
        polly.close();
    }

    private static String LEXICON_NAME = "SampleLexicon";
    public static void getLexicon(PollyClient client) {

        try {
            GetLexiconRequest getLexiconRequest = GetLexiconRequest.builder()
                .name(LEXICON_NAME).build();

            GetLexiconResponse getLexiconResult = client.getLexicon(getLexiconRequest);
            System.out.println("The name of the Lexicon is " +
getLexiconResult.lexicon().name());

        } catch (PollyException e) {
            System.err.println("Exception caught: " + e);
            System.exit(1);
        }
    }
}

```

SDK v1

```

package com.amazonaws.polly.samples;

import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.AmazonPollyClientBuilder;
import com.amazonaws.services.polly.model.GetLexiconRequest;
import com.amazonaws.services.polly.model.GetLexiconResult;

public class GetLexiconSample {

```

```
private String LEXICON_NAME = "SampleLexicon";

AmazonPolly client = AmazonPollyClientBuilder.defaultClient();

public void getLexicon() {
    GetLexiconRequest getLexiconRequest = new
    GetLexiconRequest().withName(LEXICON_NAME);

    try {
        GetLexiconResult getLexiconResult = client.getLexicon(getLexiconRequest);
        System.out.println("Lexicon: " + getLexiconResult.getLexicon());
    } catch (Exception e) {
        System.err.println("Exception caught: " + e);
    }
}
```

ListLexicons

The following Java code sample shows how to use Java-based applications to produce a list of pronunciation lexicons stored in an AWS Region.

For more information on this operation, see the reference for the [ListLexicons](#) API.

```
package com.amazonaws.polly.samples;

import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.AmazonPollyClientBuilder;
import com.amazonaws.services.polly.model.LexiconAttributes;
import com.amazonaws.services.polly.model.LexiconDescription;
import com.amazonaws.services.polly.model.ListLexiconsRequest;
import com.amazonaws.services.polly.model.ListLexiconsResult;

public class ListLexiconsSample {
    AmazonPolly client = AmazonPollyClientBuilder.defaultClient();

    public void listLexicons() {
        ListLexiconsRequest listLexiconsRequest = new ListLexiconsRequest();

        try {
            String nextToken;
            do {
```

```

        ListLexiconsResult listLexiconsResult =
client.listLexicons(listLexiconsRequest);
        nextToken = listLexiconsResult.getNextToken();
        listLexiconsRequest.setNextToken(nextToken);

        for (LexiconDescription lexiconDescription :
listLexiconsResult.getLexicons()) {
            LexiconAttributes attributes = lexiconDescription.getAttributes();
            System.out.println("Name: " + lexiconDescription.getName()
                + ", Alphabet: " + attributes.getAlphabet()
                + ", LanguageCode: " + attributes.getLanguageCode()
                + ", LastModified: " + attributes.getLastModified()
                + ", LexemesCount: " + attributes.getLexemesCount()
                + ", LexiconArn: " + attributes.getLexiconArn()
                + ", Size: " + attributes.getSize());
        }
        } while (nextToken != null);
    } catch (Exception e) {
        System.err.println("Exception caught: " + e);
    }
}
}

```

PutLexicon

The following Java code sample show how to use Java-based applications to store a pronunciation lexicon in an AWS Region.

For more information on this operation, see the reference for the [PutLexicon](#) API.

SDK v2

```

/*
    Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
    SPDX-License-Identifier: Apache-2.0
*/

package com.example.polly;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.PutLexiconRequest;

```



```

import software.amazon.awssdk.services.polly.model.PutLexiconResponse;
import software.amazon.awssdk.services.polly.model.PollyException ;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class PutLexiconSample {

    public static void main(String args[]) {

        PollyClient polly = PollyClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        putLexicon(polly) ;
        polly.close();
    }

    private static String LEXICON_CONTENT = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
        "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
        "xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-lexicon" +
        "http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" " +
        "alphabet=\"ipa\" xml:lang=\"en-US\">" +
        "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>" +
        "</lexicon>";

    private static String LEXICON_NAME = "SampleLexicon";
    public static void putLexicon(PollyClient client) {

        try {
            PutLexiconRequest putLexiconRequest = PutLexiconRequest.builder()
                .name(LEXICON_NAME).content(LEXICON_CONTENT).build();

            PutLexiconResponse putLexiconResult = client.putLexicon(putLexiconRequest);

        } catch (PollyException e) {
            System.err.println("Exception caught: " + e);
        }
    }
}

```

```

        System.exit(1);
    }
}
}

```

SDK v1

```

package com.amazonaws.polly.samples;

import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.AmazonPollyClientBuilder;
import com.amazonaws.services.polly.model.PutLexiconRequest;

public class PutLexiconSample {
    AmazonPolly client = AmazonPollyClientBuilder.defaultClient();

    private String LEXICON_CONTENT = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\" +
        "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \" \" +
        \"xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-lexicon http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" \" \" +
        \"alphabet=\"ipa\" xml:lang=\"en-US\">\" +
        "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>\" +
        "</lexicon>";
    private String LEXICON_NAME = "SampleLexicon";

    public void putLexicon() {
        PutLexiconRequest putLexiconRequest = new PutLexiconRequest()
            .withContent(LEXICON_CONTENT)
            .withName(LEXICON_NAME);

        try {
            client.putLexicon(putLexiconRequest);
        } catch (Exception e) {
            System.err.println("Exception caught: " + e);
        }
    }
}

```

StartSpeechSynthesisTask

The following Java code sample show how to use Java-based applications to synthesize a long speech (up to 100,000 billed characters) and store it directly in an Amazon S3 bucket.

For more information, see the reference for [StartSpeechSynthesisTask](#) API.

SDK v2

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */

package com.example.polly;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.*;

import java.time.Duration;
import org.awaitility.Durations;
import org.awaitility.Awaitility;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StartSpeechSynthesisTaskSample {

    public static void main(String args[]) {

        PollyClient polly = PollyClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        startSpeechSynthesisTask(polly) ;
    }
}
```

```

        polly.close();
    }

    private static final String PLAIN_TEXT = "This is a sample text to be
synthesized.";
    private static final String OUTPUT_FORMAT_MP3 = OutputFormat.MP3.toString();
    private static final String OUTPUT_BUCKET = "synth-books-buckets";
    private static final String SNS_TOPIC_ARN = "arn:aws:sns:eu-
west-2:123456789012:synthesize-finish-topic";
    private static final Duration SYNTHESIS_TASK_POLL_INTERVAL =
Durations.FIVE_SECONDS;
    private static final Duration SYNTHESIS_TASK_POLL_DELAY = Durations.TEN_SECONDS;
    private static final Duration SYNTHESIS_TASK_TIMEOUT = Durations.FIVE_MINUTES;
    public static void startSpeechSynthesisTask(PollyClient client) {

        try {
            StartSpeechSynthesisTaskRequest startSpeechSynthesisTaskRequest =
StartSpeechSynthesisTaskRequest.builder()

.outputFormat(OUTPUT_FORMAT_MP3).text(PLAIN_TEXT).textType(TextType.TEXT)

.voiceId(VoiceId.AMY).outputS3BucketName(OUTPUT_BUCKET).snsTopicArn(SNS_TOPIC_ARN)
.engine("neural").build();

            StartSpeechSynthesisTaskResponse startSpeechSynthesisTaskResponse =
                client.startSpeechSynthesisTask(startSpeechSynthesisTaskRequest);
            String taskId = startSpeechSynthesisTaskResponse.synthesisTask().taskId();

            Awaitility.await().with()
                .pollInterval(SYNTHESIS_TASK_POLL_INTERVAL)
                .pollDelay(SYNTHESIS_TASK_POLL_DELAY)
                .atMost(SYNTHESIS_TASK_TIMEOUT)
                .until(
                    () -> getSynthesisTaskStatus(client,
taskId).equals(TaskStatus.COMPLETED.toString())
                );

        } catch (PollyException e) {
            System.err.println("Exception caught: " + e);
            System.exit(1);
        }
    }

    private static String getSynthesisTaskStatus(PollyClient client, String taskId) {

```

```

        GetSpeechSynthesisTaskRequest getSpeechSynthesisTaskRequest =
        GetSpeechSynthesisTaskRequest.builder()
            .taskId(taskId).build();
        GetSpeechSynthesisTaskResponse result =
        client.getSpeechSynthesisTask(getSpeechSynthesisTaskRequest);
        return result.synthesisTask().taskStatusAsString();
    }
}

```

SDK v1

```

package com.amazonaws.parrot.service.tests.speech.task;

import com.amazonaws.parrot.service.tests.AbstractParrotServiceTest;
import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.model.*;
import org.awaitility.Duration;

import java.util.concurrent.TimeUnit;

import static org.awaitility.Awaitility.await;

public class StartSpeechSynthesisTaskSample {

    private static final int SYNTHESIS_TASK_TIMEOUT_SECONDS = 300;
    private static final AmazonPolly AMAZON_POLLY_CLIENT =
    AmazonPollyClientBuilder.defaultClient();
    private static final String PLAIN_TEXT = "This is a sample text to be
    synthesized.";
    private static final String OUTPUT_FORMAT_MP3 = OutputFormat.Mp3.toString();
    private static final String OUTPUT_BUCKET = "synth-books-buckets";
    private static final String SNS_TOPIC_ARN = "arn:aws:sns:eu-
    west-2:123456789012:synthesize-finish-topic";
    private static final Duration SYNTHESIS_TASK_POLL_INTERVAL = Duration.FIVE_SECONDS;
    private static final Duration SYNTHESIS_TASK_POLL_DELAY = Duration.TEN_SECONDS;

    public static void main(String... args) {
        StartSpeechSynthesisTaskRequest request = new StartSpeechSynthesisTaskRequest()
            .withOutputFormat(OUTPUT_FORMAT_MP3)
            .withText(PLAIN_TEXT)
            .withTextType(TextType.Text)
            .withVoiceId(VoiceId.Amy)
    }
}

```

```

        .withOutputS3BucketName(OUTPUT_BUCKET)
        .withSnsTopicArn(SNS_TOPIC_ARN)
        .withEngine("neural");

    StartSpeechSynthesisTaskResult result =
    AMAZON_POLLY_CLIENT.startSpeechSynthesisTask(request);
    String taskId = result.getSynthesisTask().getTaskId();

    await().with()
        .pollInterval(SYNTHESIS_TASK_POLL_INTERVAL)
        .pollDelay(SYNTHESIS_TASK_POLL_DELAY)
        .atMost(SYNTHESIS_TASK_TIMEOUT_SECONDS, TimeUnit.SECONDS)
        .until(
            () ->
            getSynthesisTaskStatus(taskId).equals(TaskStatus.Completed.toString())
        );
    }

    private static SynthesisTask getSynthesisTask(String taskId) {
        GetSpeechSynthesisTaskRequest getSpeechSynthesisTaskRequest = new
        GetSpeechSynthesisTaskRequest()
            .withTaskId(taskId);
        GetSpeechSynthesisTaskResult result
        =AMAZON_POLLY_CLIENT.getSpeechSynthesisTask(getSpeechSynthesisTaskRequest);
        return result.getSynthesisTask();
    }

    private static String getSynthesisTaskStatus(String taskId) {
        GetSpeechSynthesisTaskRequest getSpeechSynthesisTaskRequest = new
        GetSpeechSynthesisTaskRequest()
            .withTaskId(taskId);
        GetSpeechSynthesisTaskResult result
        =AMAZON_POLLY_CLIENT.getSpeechSynthesisTask(getSpeechSynthesisTaskRequest);
        return result.getSynthesisTask().getTaskStatus();
    }
}

```

Speech Marks

The following code sample shows how to use Java-based applications to synthesize speech marks for input text. This functionality uses the SynthesizeSpeech API.

For more information on this functionality, see [Speech marks](#).

For more information on the API, see the reference for [SynthesizeSpeech](#) API.

SDK v2

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */

package com.example.polly;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.*;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;

/**
 * Before running this Java V2 code example, set up your development environment,
 * including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class SpeechMarksSample {

    public static void main(String args[]) {

        PollyClient polly = PollyClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create())
            .build();

        speechMarksSample(polly) ;
        polly.close();
    }
}
```

```

private static final String OUTPUT_FILE = "./speechMarks.json";
public static void speechMarksSample(PollyClient client) {

    try {
        SynthesizeSpeechRequest speechMarksSampleRequest =
SynthesizeSpeechRequest.builder()
            .outputFormat(OutputFormat.JSON)
            .speechMarkTypes(SpeechMarkType.VISEME, SpeechMarkType.WORD)
            .voiceId(VoiceId.JOANNA)
            .text("This is a sample text to be synthesized")
            .build();
        try (FileOutputStream outputStream = new FileOutputStream(new
File(OUTPUT_FILE))) {
            ResponseInputStream<SynthesizeSpeechResponse> synthesizeSpeechResponse
= client
                .synthesizeSpeech(speechMarksSampleRequest);
            byte[] buffer = new byte[2 * 1024];
            int readBytes;

            try (InputStream in = synthesizeSpeechResponse){
                while ((readBytes = in.read(buffer)) > 0) {
                    outputStream.write(buffer, 0, readBytes);
                }
            }
        } catch (Exception e) {
            System.err.println("Exception caught: " + e);
        }

        } catch (PollyException e) {
            System.err.println("Exception caught: " + e);
            System.exit(1);
        }
    }
}

```

SDK v1

```

package com.amazonaws.polly.samples;

import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.AmazonPollyClientBuilder;
import com.amazonaws.services.polly.model.OutputFormat;

```



```

import com.amazonaws.services.polly.model.SpeechMarkType;
import com.amazonaws.services.polly.model.SynthesizeSpeechRequest;
import com.amazonaws.services.polly.model.SynthesizeSpeechResult;
import com.amazonaws.services.polly.model.VoiceId;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;

public class SynthesizeSpeechMarksSample {
    AmazonPolly client = AmazonPollyClientBuilder.defaultClient();

    public void synthesizeSpeechMarks() {
        String outputFileName = "/tmp/speechMarks.json";

        SynthesizeSpeechRequest synthesizeSpeechRequest = new SynthesizeSpeechRequest()
            .withOutputFormat(OutputFormat.Json)
            .withSpeechMarkTypes(SpeechMarkType.Viseme, SpeechMarkType.Word)
            .withVoiceId(VoiceId.Joanna)
            .withText("This is a sample text to be synthesized.");

        try (FileOutputStream outputStream = new FileOutputStream(new
File(outputFileName))) {
            SynthesizeSpeechResult synthesizeSpeechResult =
client.synthesizeSpeech(synthesizeSpeechRequest);
            byte[] buffer = new byte[2 * 1024];
            int readBytes;

            try (InputStream in = synthesizeSpeechResult.getAudioStream()){
                while ((readBytes = in.read(buffer)) > 0) {
                    outputStream.write(buffer, 0, readBytes);
                }
            }
        } catch (Exception e) {
            System.err.println("Exception caught: " + e);
        }
    }
}

```

SynthesizeSpeech

The following Java code sample show how to use Java-based applications to synthesize speech with shorter texts for near-real time processing.

For more information, see the reference for [SynthesizeSpeech](#) API.

```
package com.amazonaws.polly.samples;

import com.amazonaws.services.polly.AmazonPolly;
import com.amazonaws.services.polly.AmazonPollyClientBuilder;
import com.amazonaws.services.polly.model.OutputFormat;
import com.amazonaws.services.polly.model.SynthesizeSpeechRequest;
import com.amazonaws.services.polly.model.SynthesizeSpeechResult;
import com.amazonaws.services.polly.model.VoiceId;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;

public class SynthesizeSpeechSample {
    AmazonPolly client = AmazonPollyClientBuilder.defaultClient();

    public void synthesizeSpeech() {
        String outputFileName = "/tmp/speech.mp3";

        SynthesizeSpeechRequest synthesizeSpeechRequest = new SynthesizeSpeechRequest()
            .withOutputFormat(OutputFormat.Mp3)
            .withVoiceId(VoiceId.Joanna)
            .withText("This is a sample text to be synthesized.")
            .withEngine("neural");

        try (FileOutputStream outputStream = new FileOutputStream(new
File(outputFileName))) {
            SynthesizeSpeechResult synthesizeSpeechResult =
client.synthesizeSpeech(synthesizeSpeechRequest);
            byte[] buffer = new byte[2 * 1024];
            int readBytes;

            try (InputStream in = synthesizeSpeechResult.getAudioStream()){
                while ((readBytes = in.read(buffer)) > 0) {
                    outputStream.write(buffer, 0, readBytes);
                }
            }
        } catch (Exception e) {
            System.err.println("Exception caught: " + e);
        }
    }
}
```

```
}
```

Python samples

The following code samples show how to use Python (boto3)-based applications to accomplish various tasks with Amazon Polly. These samples are not intended to be full examples, but can be included in larger Python applications that use the [AWS SDK for Python \(Boto\)](#).

Code Snippets

- [DeleteLexicon](#)
- [GetLexicon](#)
- [ListLexicon](#)
- [PutLexicon](#)
- [StartSpeechSynthesisTask](#)
- [SynthesizeSpeech](#)

DeleteLexicon

The following Python code example uses the AWS SDK for Python (Boto) to delete a lexicon in the region specified in your local AWS configuration. The example deletes only the specified lexicon. It asks you to confirm that you want to proceed before actually deleting the lexicon.

The following code example uses default credentials stored in the AWS SDK configuration file. For information about creating the configuration file, see [Setting up the AWS CLI](#).

For more information on this operation, see the reference for the [DeleteLexicon](#) API.

```
from argparse import ArgumentParser
from sys import version_info

from boto3 import Session
from botocore.exceptions import BotoCoreError, ClientError

# Define and parse the command line arguments
cli = ArgumentParser(description="DeleteLexicon example")
cli.add_argument("name", type=str, metavar="LEXICON_NAME")
arguments = cli.parse_args()
```

```
# Create a client using the credentials and region defined in the adminuser
# section of the AWS credentials and configuration files
session = Session(profile_name="adminuser")
polly = session.client("polly")

# Request confirmation
prompt = input if version_info >= (3, 0) else raw_input
proceed = prompt((u"This will delete the \"{0}\" lexicon,"
                  " do you want to proceed? [y,n]: ").format(arguments.name))

if proceed in ("y", "Y"):
    print(u"Deleting {0}...".format(arguments.name))

    try:
        # Request deletion of a lexicon by name
        response = polly.delete_lexicon(Name=arguments.name)
    except (BotoCoreError, ClientError) as error:
        # The service returned an error, exit gracefully
        cli.error(error)

    print("Done.")
else:
    print("Cancelled.")
```

GetLexicon

The following Python code uses the AWS SDK for Python (Boto) to retrieve all lexicons stored in an AWS Region. The example accepts a lexicon name as a command line parameter and fetches that lexicon only, printing out the tmp path where it has been saved locally.

The following code example uses default credentials stored in the AWS SDK configuration file. For information about creating the configuration file, see [Setting up the AWS CLI](#).

For more information on this operation, see the reference for the [GetLexicon](#) API.

```
from argparse import ArgumentParser
from os import path
from tempfile import gettempdir

from boto3 import Session
from botocore.exceptions import BotoCoreError, ClientError
```

```
# Define and parse the command line arguments
cli = ArgumentParser(description="GetLexicon example")
cli.add_argument("name", type=str, metavar="LEXICON_NAME")
arguments = cli.parse_args()

# Create a client using the credentials and region defined in the adminuser
# section of the AWS credentials and configuration files
session = Session(profile_name="adminuser")
polly = session.client("polly")

print(u"Fetching {0}...".format(arguments.name))

try:
    # Fetch lexicon by name
    response = polly.get_lexicon(Name=arguments.name)
except (BotoCoreError, ClientError) as error:
    # The service returned an error, exit gracefully
    cli.error(error)

# Get the lexicon data from the response
lexicon = response.get("Lexicon", {})

# Access the lexicon's content
if "Content" in lexicon:
    output = path.join(gettempdir(), u"%s.pls" % arguments.name)
    print(u"Saving to %s..." % output)

    try:
        # Save the lexicon contents to a local file
        with open(output, "w") as pls_file:
            pls_file.write(lexicon["Content"])
    except IOError as error:
        # Could not write to file, exit gracefully
        cli.error(error)
else:
    # The response didn't contain lexicon data, exit gracefully
    cli.error("Could not fetch lexicons contents")

print("Done.")
```

ListLexicon

The following Python code example uses the AWS SDK for Python (Boto) to list the lexicons in your account in the region specified in your local AWS configuration. For information about creating the configuration file, see [Setting up the AWS CLI](#).

For more information on this operation, see the reference for the [ListLexicons](#) API.

```
import sys

from boto3 import Session
from botocore.exceptions import BotoCoreError, ClientError

# Create a client using the credentials and region defined in the adminuser
# section of the AWS credentials and configuration files
session = Session(profile_name="adminuser")
polly = session.client("polly")

try:
    # Request the list of available lexicons
    response = polly.list_lexicons()
except (BotoCoreError, ClientError) as error:
    # The service returned an error, exit gracefully
    print(error)
    sys.exit(-1)

# Get the list of lexicons in the response
lexicons = response.get("Lexicons", [])
print("{0} lexicon(s) found".format(len(lexicons)))

# Output a formatted list of lexicons with some of the attributes
for lexicon in lexicons:
    print((u" - {Name} ({Attributes[LanguageCode]}), "
          "{Attributes[LexemesCount]} lexeme(s)").format(**lexicon))
```

PutLexicon

The following code sample show how to use Python (boto3)-based applications to store a pronunciation lexicon in an AWS Region.

For more information on this operation, see the reference for the [PutLexicon](#) API.

Note the following:

- You need to update the code by providing a local lexicon file name and a stored lexicon name.
- The example assumes you have lexicon files created in a subdirectory called `pls`. You need to update the path as appropriate.

The following code example uses default credentials stored in the AWS SDK configuration file. For information about creating the configuration file, see [Setting up the AWS CLI](#).

For more information on this operation, see the reference for the [PutLexicon](#) API.

```
from argparse import ArgumentParser

from boto3 import Session
from botocore.exceptions import BotoCoreError, ClientError

# Define and parse the command line arguments
cli = ArgumentParser(description="PutLexicon example")
cli.add_argument("path", type=str, metavar="FILE_PATH")
cli.add_argument("-n", "--name", type=str, required=True,
                  metavar="LEXICON_NAME", dest="name")
arguments = cli.parse_args()

# Create a client using the credentials and region defined in the adminuser
# section of the AWS credentials and configuration files
session = Session(profile_name="adminuser")
polly = session.client("polly")

# Open the PLS lexicon file for reading
try:
    with open(arguments.path, "r") as lexicon_file:
        # Read the pls file contents
        lexicon_data = lexicon_file.read()

        # Store the PLS lexicon on the service.
        # If a lexicon with that name already exists,
        # its contents will be updated
        response = polly.put_lexicon(Name=arguments.name,
                                     Content=lexicon_data)
except (IOError, BotoCoreError, ClientError) as error:
    # Could not open/read the file or the service returned an error,
    # exit gracefully
    cli.error(error)
```

```
print(u"The \"{0}\" lexicon is now available for use.".format(arguments.name))
```

StartSpeechSynthesisTask

The following Python code example uses the AWS SDK for Python (Boto) to list the lexicons in your account in the region specified in your local AWS configuration. For information about creating the configuration file, see [Setting up the AWS CLI](#).

For more information, see the reference for [StartSpeechSynthesisTask](#) API.

```
import boto3
import time

polly_client = boto3.Session(
    aws_access_key_id='',
    aws_secret_access_key='',
    region_name='eu-west-2').client('polly')

response = polly_client.start_speech_synthesis_task(VoiceId='Joanna',
    OutputS3BucketName='synth-books-buckets',
    OutputS3KeyPrefix='key',
    OutputFormat='mp3',
    Text='This is a sample text to be synthesized.',
    Engine='neural')

taskId = response['SynthesisTask']['TaskId']

print( "Task id is {} ".format(taskId))

task_status = polly_client.get_speech_synthesis_task(TaskId = taskId)

print(task_status)
```

SynthesizeSpeech

The following Python code example uses the AWS SDK for Python (Boto) to synthesize speech with shorter texts for near real-time processing. For more information, see the reference for the [SynthesizeSpeech](#) operation.

This example uses a short string of plain text. You can use SSML text for more control over the output. For more information, see [Generating speech from SSML documents](#).

```
import boto3

polly_client = boto3.Session(
    aws_access_key_id=,
    aws_secret_access_key=,
    region_name='us-west-2').client('polly')

response = polly_client.synthesize_speech(VoiceId='Joanna',
    OutputFormat='mp3',
    Text = 'This is a sample text to be synthesized.',
    Engine = 'neural')

file = open('speech.mp3', 'wb')
file.write(response['AudioStream'].read())
file.close()
```

Java example

This example shows how to use Amazon Polly to stream speech from a Java-based application. The example uses the [AWS SDK for Java](#) to read the specified text using a voice selected from a list.

The code shown covers major tasks, but does only minimal error checking. If Amazon Polly encounters an error, the application terminates.

To run this example application, you need the following:

- Java 8 Java Development Kit (JDK)
- [AWS SDK for Java](#)
- [Apache Maven](#)

To test the application

1. Ensure that the JAVA_HOME environment variable is set for the JDK.

For example, if you installed JDK 1.8.0_121 on Windows at C:\Program Files\Java\jdk1.8.0_121, you would type the following at the command prompt:

```
set JAVA_HOME=""C:\Program Files\Java\jdk1.8.0_121""
```

If you installed JDK 1.8.0_121 in Linux at `/usr/lib/jvm/java8-openjdk-amd64`, you would type the following at the command prompt:

```
export JAVA_HOME=/usr/lib/jvm/java8-openjdk-amd64
```

2. Set the Maven environment variables to run Maven from the command line.

For example, if you installed Maven 3.3.9 on Windows at `C:\Program Files\apache-maven-3.3.9`, you would type the following:

```
set M2_HOME=""C:\Program Files\apache-maven-3.3.9""
set M2=%M2_HOME%\bin
set PATH=%M2%;%PATH%
```

If you installed Maven 3.3.9 on Linux at `/home/ec2-user/opt/apache-maven-3.3.9`, you would type the following:

```
export M2_HOME=/home/ec2-user/opt/apache-maven-3.3.9
export M2=$M2_HOME/bin
export PATH=$M2:$PATH
```

3. Create a new directory called `polly-java-demo`.
4. In the `polly-java-demo` directory, create a new file called `pom.xml`, and paste the following code into it:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws.polly</groupId>
  <artifactId>java-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-polly -->
    <dependency>
```

```
<groupId>com.amazonaws</groupId>
<artifactId>aws-java-sdk-polly</artifactId>
<version>1.11.77</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.googlecode.soundlibs/jlayer -->
<dependency>
  <groupId>com.googlecode.soundlibs</groupId>
  <artifactId>jlayer</artifactId>
  <version>1.0.1-1</version>
</dependency>

</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>1.2.1</version>
      <executions>
        <execution>
          <goals>
            <goal>java</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <mainClass>com.amazonaws.demos.polly.PollyDemo</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

5. Create a new directory called `polly` at `src/main/java/com/amazonaws/demos`.
6. In the `polly` directory, create a new Java source file called `PollyDemo.java`, and paste in the following code:

```
package com.amazonaws.demos.polly;

import java.io.IOException;
import java.io.InputStream;
```

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.polly.AmazonPollyClient;
import com.amazonaws.services.polly.model.DescribeVoicesRequest;
import com.amazonaws.services.polly.model.DescribeVoicesResult;
import com.amazonaws.services.polly.model.OutputFormat;
import com.amazonaws.services.polly.model.SynthesizeSpeechRequest;
import com.amazonaws.services.polly.model.SynthesizeSpeechResult;
import com.amazonaws.services.polly.model.Voice;

import javax.swing.plaf.basic.AdvancedPlayer;
import javax.swing.plaf.basic.PlaybackEvent;
import javax.swing.plaf.basic.PlaybackListener;

public class PollyDemo {

    private final AmazonPollyClient polly;
    private final Voice voice;
    private static final String SAMPLE = "Congratulations. You have successfully built
this working demo
of Amazon Polly in Java. Have fun building voice enabled apps with Amazon Polly
(that's me!), and always
look at the AWS website for tips and tricks on using Amazon Polly and other great
services from AWS";

    public PollyDemo(Region region) {
        // create an Amazon Polly client in a specific region
        polly = new AmazonPollyClient(new DefaultAWSCredentialsProviderChain(),
            new ClientConfiguration());
        polly.setRegion(region);
        // Create describe voices request.
        DescribeVoicesRequest describeVoicesRequest = new DescribeVoicesRequest();

        // Synchronously ask Amazon Polly to describe available TTS voices.
        DescribeVoicesResult describeVoicesResult =
            polly.describeVoices(describeVoicesRequest);
        voice = describeVoicesResult.getVoices().get(0);
    }

    public InputStream synthesize(String text, OutputFormat format) throws IOException
    {
        SynthesizeSpeechRequest synthReq =
```

```

    new SynthesizeSpeechRequest().withText(text).withVoiceId(voice.getId())
        .withOutputFormat(format).withEngine("neural");
    SynthesizeSpeechResult synthRes = polly.synthesizeSpeech(synthReq);

    return synthRes.getAudioStream();
}

public static void main(String args[]) throws Exception {
    //create the test class
    PollyDemo helloWorld = new PollyDemo(Region.getRegion(Regions.US_EAST_1));
    //get the audio stream
    InputStream speechStream = helloWorld.synthesize(SAMPLE, OutputFormat.Mp3);

    //create an MP3 player
    AdvancedPlayer player = new AdvancedPlayer(speechStream,
        javazoom.jl.player.FactoryRegistry.systemRegistry().createAudioDevice());

    player.setPlaybackListener(new PlaybackListener() {
        @Override
        public void playbackStarted(PlaybackEvent evt) {
            System.out.println("Playback started");
            System.out.println(SAMPLE);
        }

        @Override
        public void playbackFinished(PlaybackEvent evt) {
            System.out.println("Playback finished");
        }
    });

    // play it!
    player.play();
}
}

```

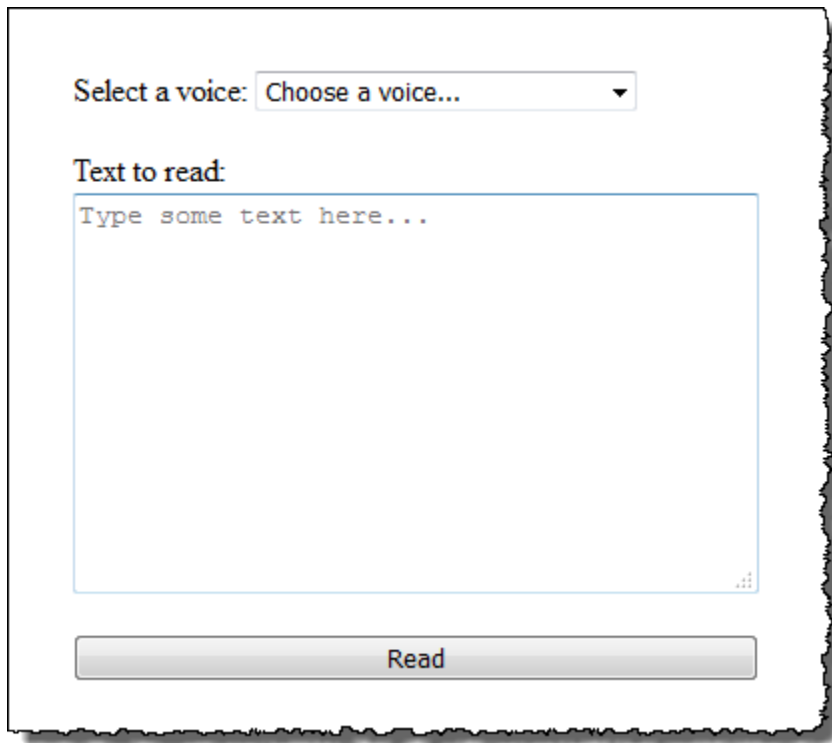
7. Return to the polly-java-demo directory to clean, compile, and execute the demo:

```
mvn clean compile exec:java
```

Python example (HTML5 Client and Python Server)

This example application consists of the following:

- An HTTP 1.1 server using the HTTP chunked transfer coding (see [Chunked Transfer Coding](#))
- A simple HTML5 user interface that interacts with the HTTP 1.1 server (shown below):



Select a voice: Choose a voice...

Text to read:

Type some text here...

Read

The goal of this example is to show how to use Amazon Polly to stream speech from a browser-based HTML5 application. Consuming the audio stream produced by Amazon Polly as the text gets synthesized is the recommended approach for use cases where responsiveness is an important factor (for example, dialog systems, screen readers, etc.).

To run this example application you need the following:

- Web browser compliant with the HTML5 and EcmaScript5 standards (for example, Chrome 23.0 or higher, Firefox 21.0 or higher, Internet Explorer 9.0, or higher)
- Python version greater than 3.0

To test the application

1. Save the server code as `server.py`. For the code, see [Python example: Python Server Code \(server.py\)](#).
2. Save the HTML5 client code as `index.html`. For the code, see [Python example: HTML5 User Interface \(index.html\)](#).
3. Run the following command from the path where you saved `server.py` to start the application (on some systems you might need to use `python3` instead of `python` when running the command).

```
$ python server.py
```

After the application starts, a URL appears on the terminal.

4. Open the URL shown in the terminal in a web browser.

You can pass the address and port for the application server to use as a parameter to `server.py`. For more information, run `python server.py -h`.

5. To listen to speech, choose a voice from the list, type some text, and then choose **Read**. The speech starts playing as soon as Amazon Polly transfers the first usable chunk of audio data.
6. To stop the Python server when you're finished testing the application, press Ctrl+C in the terminal where the server is running.

Note

The server creates a Boto3 client using the AWS SDK for Python (Boto). The client uses the credentials stored in the AWS config file on your computer to sign and authenticate the requests to Amazon Polly. For more information on how to create the AWS config file and store credentials, see [Configuring the AWS Command Line Interface](#) in the *AWS Command Line Interface User Guide*.

Python example: HTML5 User Interface (index.html)

This section provides the code for the HTML5 client described in [Python example \(HTML5 Client and Python Server\)](#).

```
<html>

<head>
  <title>Text-to-Speech Example Application</title>
  <script>
    /*
      * This sample code requires a web browser with support for both the
      * HTML5 and ECMAScript 5 standards; the following is a non-comprehensive
      * list of compliant browsers and their minimum version:
      *
      * - Chrome 23.0+
      * - Firefox 21.0+
      * - Internet Explorer 9.0+
      * - Edge 12.0+
      * - Opera 15.0+
      * - Safari 6.1+
      * - Android (stock web browser) 4.4+
      * - Chrome for Android 51.0+
      * - Firefox for Android 48.0+
      * - Opera Mobile 37.0+
      * - iOS (Safari Mobile and Chrome) 3.2+
      * - Internet Explorer Mobile 10.0+
      * - Blackberry Browser 10.0+
      */

    // Mapping of the OutputFormat parameter of the SynthesizeSpeech API
    // and the audio format strings understood by the browser
    var AUDIO_FORMATS = {
      'ogg_vorbis': 'audio/ogg',
      'mp3': 'audio/mpeg',
      'pcm': 'audio/wave; codecs=1',
      'mulaw': 'audio/mulaw',
      'alaw': 'audio/alaw'
    };

    /**
     * Handles fetching JSON over HTTP
     */
    function fetchJSON(method, url, onSuccess, onError) {
      var request = new XMLHttpRequest();
      request.open(method, url, true);
      request.onload = function () {
        // If loading is complete
```



```

        if (request.readyState === 4) {
            // if the request was successful
            if (request.status === 200) {
                var data;

                // Parse the JSON in the response
                try {
                    data = JSON.parse(request.responseText);
                } catch (error) {
                    onError(request.status, error.toString());
                }

                onSuccess(data);
            } else {
                onError(request.status, request.responseText)
            }
        }
    };

    request.send();
}

/**
 * Returns a list of audio formats supported by the browser
 */
function getSupportedAudioFormats(player) {
    return Object.keys(AUDIO_FORMATS)
        .filter(function (format) {
            var supported = player.canPlayType(AUDIO_FORMATS[format]);
            return supported === 'probably' || supported === 'maybe';
        });
}

// Initialize the application when the DOM is loaded and ready to be
// manipulated
document.addEventListener("DOMContentLoaded", function () {
    var input = document.getElementById('input'),
        voiceMenu = document.getElementById('voice'),
        text = document.getElementById('text'),
        player = document.getElementById('player'),
        submit = document.getElementById('submit'),
        supportedFormats = getSupportedAudioFormats(player);

    // Display a message and don't allow submitting the form if the

```

```

// browser doesn't support any of the available audio formats
if (supportedFormats.length === 0) {
    submit.disabled = true;
    alert('The web browser in use does not support any of the' +
        ' available audio formats. Please try with a different' +
        ' one.');
```

}

```

// Play the audio stream when the form is submitted successfully
input.addEventListener('submit', function (event) {
    // Validate the fields in the form, display a message if
    // unexpected values are encountered
    if (voiceMenu.selectedIndex <= 0 || text.value.length === 0) {
        alert('Please fill in all the fields.');
```

} else {

```

        var selectedVoice = voiceMenu
                                .options[voiceMenu.selectedIndex]
                                .value;

        // Point the player to the streaming server
        player.src = '/read?voiceId=' +
            encodeURIComponent(selectedVoice) +
            '&text=' + encodeURIComponent(text.value) +
            '&outputFormat=' + supportedFormats[0];
        player.play();
    }

    // Stop the form from submitting,
    // Submitting the form is allowed only if the browser doesn't
    // support Javascript to ensure functionality in such a case
    event.preventDefault();
});

// Load the list of available voices and display them in a menu
fetchJSON('GET', '/voices',
    // If the request succeeds
    function (voices) {
        var container = document.createDocumentFragment();

        // Build the list of options for the menu
        voices.forEach(function (voice) {
            var option = document.createElement('option');
            option.value = voice['Id'];
            option.innerHTML = voice['Name'] + ' (' +
```

```

        voice['Gender'] + ', ' +
        voice['LanguageName'] + ')';
    container.appendChild(option);
});

    // Add the options to the menu and enable the form field
    voiceMenu.appendChild(container);
    voiceMenu.disabled = false;
},
// If the request fails
function (status, response) {
    // Display a message in case loading data from the server
    // fails
    alert(status + ' - ' + response);
});
});

</script>
<style>
    #input {
        min-width: 100px;
        max-width: 600px;
        margin: 0 auto;
        padding: 50px;
    }

    #input div {
        margin-bottom: 20px;
    }

    #text {
        width: 100%;
        height: 200px;
        display: block;
    }

    #submit {
        width: 100%;
    }
</style>
</head>

<body>
    <form id="input" method="GET" action="/read">

```

```
<div>
  <label for="voice">Select a voice:</label>
  <select id="voice" name="voiceId" disabled>
    <option value="">Choose a voice...</option>
  </select>
</div>
<div>
  <label for="text">Text to read:</label>
  <textarea id="text" maxlength="1000" minlength="1" name="text"
    placeholder="Type some text here..."></textarea>
</div>
<input type="submit" value="Read" id="submit" />
</form>
<audio id="player"></audio>
</body>

</html>
```

Python example: Python Server Code (server.py)

This section provides the code for the Python server described in [Python example \(HTML5 Client and Python Server\)](#).

```
"""
```

Example Python 2.7+/3.3+ Application

This application consists of a HTTP 1.1 server using the HTTP chunked transfer coding (<https://tools.ietf.org/html/rfc2616#section-3.6.1>) and a minimal HTML5 user interface that interacts with it.

The goal of this example is to start streaming the speech to the client (the HTML5 web UI) as soon as the first consumable chunk of speech is returned in order to start playing the audio as soon as possible.

For use cases where low latency and responsiveness are strong requirements, this is the recommended approach.

The service documentation contains examples for non-streaming use cases where waiting for the speech synthesis to complete and fetching the whole audio stream at once are an option.

To test the application, run 'python server.py' and then open the URL displayed in the terminal in a web browser (see index.html for a list of supported browsers). The address and port for the server can be passed as

```

parameters to server.py. For more information, run: 'python server.py -h'
"""
from argparse import ArgumentParser
from collections import namedtuple
from contextlib import closing
from io import BytesIO
from json import dumps as json_encode
import os
import sys

if sys.version_info >= (3, 0):
    from http.server import BaseHTTPRequestHandler, HTTPServer
    from socketserver import ThreadingMixIn
    from urllib.parse import parse_qs
else:
    from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
    from SocketServer import ThreadingMixIn
    from urlparse import parse_qs

from boto3 import Session
from botocore.exceptions import BotoCoreError, ClientError

ResponseStatus = namedtuple("HTTPStatus",
                           ["code", "message"])

ResponseData = namedtuple("ResponseData",
                           ["status", "content_type", "data_stream"])

# Mapping the output format used in the client to the content type for the
# response
AUDIO_FORMATS = {"ogg_vorbis": "audio/ogg",
                  "mp3": "audio/mpeg",
                  "pcm": "audio/wave; codecs=1",
                  "mulaw": "audio/mulaw",
                  "alaw": "audio/alaw"}

CHUNK_SIZE = 1024
HTTP_STATUS = {"OK": ResponseStatus(code=200, message="OK"),
               "BAD_REQUEST": ResponseStatus(code=400, message="Bad request"),
               "NOT_FOUND": ResponseStatus(code=404, message="Not found"),
               "INTERNAL_SERVER_ERROR": ResponseStatus(code=500, message="Internal
server error")}
PROTOCOL = "http"
ROUTE_INDEX = "/index.html"
ROUTE_VOICES = "/voices"

```

```

ROUTE_READ = "/read"

# Create a client using the credentials and region defined in the adminuser
# section of the AWS credentials and configuration files
session = Session(profile_name="adminuser")
polly = session.client("polly")

class HTTPStatusError(Exception):
    """Exception wrapping a value from http.server.HTTPStatus"""

    def __init__(self, status, description=None):
        """
        Constructs an error instance from a tuple of
        (code, message, description), see http.server.HTTPStatus
        """
        super(HTTPStatusError, self).__init__()
        self.code = status.code
        self.message = status.message
        self.explain = description

class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):
    """An HTTP Server that handle each request in a new thread"""
    daemon_threads = True

class ChunkedHTTPRequestHandler(BaseHTTPRequestHandler):
    """HTTP 1.1 Chunked encoding request handler"""
    # Use HTTP 1.1 as 1.0 doesn't support chunked encoding
    protocol_version = "HTTP/1.1"

    def query_get(self, queryData, key, default=""):
        """Helper for getting values from a pre-parsed query string"""
        return queryData.get(key, [default])[0]

    def do_GET(self):
        """Handles GET requests"""

        # Extract values from the query string
        path, _, query_string = self.path.partition('?')
        query = parse_qs(query_string)

```

```

response = None

print(u"[START]: Received GET for %s with query: %s" % (path, query))

try:
    # Handle the possible request paths
    if path == ROUTE_INDEX:
        response = self.route_index(path, query)
    elif path == ROUTE_VOICES:
        response = self.route_voices(path, query)
    elif path == ROUTE_READ:
        response = self.route_read(path, query)
    else:
        response = self.route_not_found(path, query)

    self.send_headers(response.status, response.content_type)
    self.stream_data(response.data_stream)

except HTTPStatusError as err:
    # Respond with an error and log debug
    # information
    if sys.version_info >= (3, 0):
        self.send_error(err.code, err.message, err.explain)
    else:
        self.send_error(err.code, err.message)

    self.log_error(u"%s %s %s - [%d] %s", self.client_address[0],
                  self.command, self.path, err.code, err.explain)

print("[END]")

def route_not_found(self, path, query):
    """Handles routing for unexpected paths"""
    raise HTTPStatusError(HTTP_STATUS["NOT_FOUND"], "Page not found")

def route_index(self, path, query):
    """Handles routing for the application's entry point"""
    try:
        return ResponseData(status=HTTP_STATUS["OK"], content_type="text_html",
                            # Open a binary stream for reading the index
                            # HTML file
                            data_stream=open(os.path.join(sys.path[0],
                                                            path[1:]), "rb"))
    except IOError as err:

```

```

        # Couldn't open the stream
        raise HTTPStatusError(HTTP_STATUS["INTERNAL_SERVER_ERROR"],
                               str(err))

def route_voices(self, path, query):
    """Handles routing for listing available voices"""
    params = {}
    voices = []

    while True:
        try:
            # Request list of available voices, if a continuation token
            # was returned by the previous call then use it to continue
            # listing
            response = polly.describe_voices(**params)
        except (BotoCoreError, ClientError) as err:
            # The service returned an error
            raise HTTPStatusError(HTTP_STATUS["INTERNAL_SERVER_ERROR"],
                                   str(err))

        # Collect all the voices
        voices.extend(response.get("Voices", []))

        # If a continuation token was returned continue, stop iterating
        # otherwise
        if "NextToken" in response:
            params = {"NextToken": response["NextToken"]}
        else:
            break

    json_data = json_encode(voices)
    bytes_data = bytes(json_data, "utf-8") if sys.version_info >= (3, 0) \
        else bytes(json_data)

    return ResponseData(status=HTTP_STATUS["OK"],
                        content_type="application/json",
                        # Create a binary stream for the JSON data
                        data_stream=BytesIO(bytes_data))

def route_read(self, path, query):
    """Handles routing for reading text (speech synthesis)"""
    # Get the parameters from the query string
    text = self.query_get(query, "text")
    voiceId = self.query_get(query, "voiceId")

```



```

outputFormat = self.query_get(query, "outputFormat")

# Validate the parameters, set error flag in case of unexpected
# values
if len(text) == 0 or len(voiceId) == 0 or \
    outputFormat not in AUDIO_FORMATS:
    raise HTTPStatusError(HTTP_STATUS["BAD_REQUEST"],
                          "Wrong parameters")
else:
    try:
        # Request speech synthesis
        response = polly.synthesize_speech(Text=text,
                                           VoiceId=voiceId,
                                           OutputFormat=outputFormat,
                                           Engine="neural")

    except (BotoCoreError, ClientError) as err:
        # The service returned an error
        raise HTTPStatusError(HTTP_STATUS["INTERNAL_SERVER_ERROR"],
                              str(err))

    return ResponseData(status=HTTP_STATUS["OK"],
                       content_type=AUDIO_FORMATS[outputFormat],
                       # Access the audio stream in the response
                       data_stream=response.get("AudioStream"))

def send_headers(self, status, content_type):
    """Send out the group of headers for a successful request"""
    # Send HTTP headers
    self.send_response(status.code, status.message)
    self.send_header('Content-type', content_type)
    self.send_header('Transfer-Encoding', 'chunked')
    self.send_header('Connection', 'close')
    self.end_headers()

def stream_data(self, stream):
    """Consumes a stream in chunks to produce the response's output"""
    print("Streaming started...")

    if stream:
        # Note: Closing the stream is important as the service throttles on
        # the number of parallel connections. Here we are using
        # contextlib.closing to ensure the close method of the stream object
        # will be called automatically at the end of the with statement's
        # scope.

```

```

        with closing(stream) as managed_stream:
            # Push out the stream's content in chunks
            while True:
                data = managed_stream.read(CHUNK_SIZE)
                self.wfile.write(b"%X\r\n%s\r\n" % (len(data), data))

                # If there's no more data to read, stop streaming
                if not data:
                    break

            # Ensure any buffered output has been transmitted and close the
            # stream
            self.wfile.flush()

        print("Streaming completed.")
    else:
        # The stream passed in is empty
        self.wfile.write(b"0\r\n\r\n")
        print("Nothing to stream.")

# Define and parse the command line arguments
cli = ArgumentParser(description='Example Python Application')
cli.add_argument(
    "-p", "--port", type=int, metavar="PORT", dest="port", default=8000)
cli.add_argument(
    "--host", type=str, metavar="HOST", dest="host", default="localhost")
arguments = cli.parse_args()

# If the module is invoked directly, initialize the application
if __name__ == '__main__':
    # Create and configure the HTTP server instance
    server = ThreadedHTTPServer((arguments.host, arguments.port),
                               ChunkedHTTPRequestHandler)

    print("Starting server, use <Ctrl-C> to stop...")
    print(u"Open {0}://{1}:{2}{3} in a web browser.".format(PROTOCOL,
                                                         arguments.host,
                                                         arguments.port,
                                                         ROUTE_INDEX))

    try:
        # Listen for requests indefinitely
        server.serve_forever()
    except KeyboardInterrupt:
        # A request to terminate has been received, stop the server

```

```
print("\nShutting down...")
server.socket.close()
```

iOS example

The following example uses the iOS SDK for Amazon Polly to read the specified text using a voice selected from a list of voices.

The code shown here covers the major tasks but does not handle errors. For the complete code, see [AWS Mobile SDK for iOS Amazon Polly demo](#).

Initialize

```
// Region of Amazon Polly.
let AwsRegion = AWSRegionType.usEast1

// Cognito pool ID. Pool needs to be unauthenticated pool with
// Amazon Polly permissions.
let CognitoIdentityPoolId = "YourCognitoIdentityPoolId"

// Initialize the Amazon Cognito credentials provider.
let credentialProvider = AWSCognitoCredentialsProvider(regionType: AwsRegion,
    identityPoolId: CognitoIdentityPoolId)

// Create an audio player
var audioPlayer = AVPlayer()
```

Get List of Available Voices

```
// Use the configuration as default
AWSServiceManager.default().defaultServiceConfiguration = configuration

// Get all the voices (no parameters specified in input) from Amazon Polly
// This creates an async task.
let task = AWSPolly.default().describeVoices(AWSPollyDescribeVoicesInput())

// When the request is done, asynchronously do the following block
// (we ignore all the errors, but in a real-world scenario they need
// to be handled)
```

```
task.continue(successBlock: { (awsTask: AWSTask) -> Any? in
    // awsTask.result is an instance of AWSPollyDescribeVoicesOutput in
    // case of the "describeVoices" method
    let voices = (awsTask.result! as AWSPollyDescribeVoicesOutput).voices

    return nil
})
```

Synthesize Speech

```
// First, Amazon Polly requires an input, which we need to prepare.
// Again, we ignore the errors, however this should be handled in
// real applications. Here we are using the URL Builder Request,
// since in order to make the synthesis quicker we will pass the
// presigned URL to the system audio player.
let input = AWSPollySynthesizeSpeechURLBuilderRequest()

// Text to synthesize
input.text = "Sample text"

// We expect the output in MP3 format
input.outputFormat = AWSPollyOutputFormat.mp3

// Choose the voice ID
input.voiceId = AWSPollyVoiceId.joanna

// Create an task to synthesize speech using the given synthesis input
let builder = AWSPollySynthesizeSpeechURLBuilder.default().getPresignedURL(input)

// Request the URL for synthesis result
builder.continueOnSuccessWith(block: { (awsTask: AWSTask<NSURL>) -> Any? in
    // The result of getPresignedURL task is NSURL.
    // Again, we ignore the errors in the example.
    let url = awsTask.result!

    // Try playing the data using the system AVAudioPlayer
    self.audioPlayer.replaceCurrentItem(with: AVPlayerItem(url: url as URL))
    self.audioPlayer.play()

    return nil
})
```

Android example

The following example uses the Android SDK for Amazon Polly to read the specified text using a voice selected from a list of voices.

The code shown here covers the major tasks but does not handle errors. For the complete code, see the [AWS Mobile SDK for Android Amazon Polly demo](#).

Initialize

```
// Cognito pool ID. Pool needs to be unauthenticated pool with
// Amazon Polly permissions.
String COGNITO_POOL_ID = "YourCognitoIdentityPoolId";

// Region of Amazon Polly.
Regions MY_REGION = Regions.US_EAST_1;

// Initialize the Amazon Cognito credentials provider.
CognitoCachingCredentialsProvider credentialsProvider = new
    CognitoCachingCredentialsProvider(
        getApplicationContext(),
        COGNITO_POOL_ID,
        MY_REGION
    );

// Create a client that supports generation of presigned URLs.
AmazonPollyPresigningClient client = new
    AmazonPollyPresigningClient(credentialsProvider);
```

Get List of Available Voices

```
// Create describe voices request.
DescribeVoicesRequest describeVoicesRequest = new DescribeVoicesRequest();

// Synchronously ask Amazon Polly to describe available TTS voices.
DescribeVoicesResult describeVoicesResult =
    client.describeVoices(describeVoicesRequest);
List<Voice> voices = describeVoicesResult.getVoices();
```

Get URL for Audio Stream

```
// Create speech synthesis request.
SynthesizeSpeechPresignRequest synthesizeSpeechPresignRequest =
    new SynthesizeSpeechPresignRequest()
    // Set the text to synthesize.
    .withText("Hello world!")
    // Select voice for synthesis.
    .withVoiceId(voices.get(0).getId()) // "Joanna"
    // Set format to MP3.
    .withOutputFormat(OutputFormat.Mp3);

// Get the presigned URL for synthesized speech audio stream.
URL presignedSynthesizeSpeechUrl =
    client.getPresignedSynthesizeSpeechUrl(synthesizeSpeechPresignRequest);
```

Play Synthesized Speech

```
// Use MediaPlayer: https://developer.android.com/guide/topics/media/mediaplayer.html

// Create a media player to play the synthesized audio stream.
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);

try {
    // Set media player's data source to previously obtained URL.
    mediaPlayer.setDataSource(presignedSynthesizeSpeechUrl.toString());
} catch (IOException e) {
    Log.e(TAG, "Unable to set data source for the media player! " + e.getMessage());
}

// Prepare the MediaPlayer asynchronously (since the data source is a network stream).
mediaPlayer.prepareAsync();

// Set the callback to start the MediaPlayer when it's prepared.
mediaPlayer.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
    @Override
    public void onPrepared(MediaPlayer mp) {
        mp.start();
    }
});

// Set the callback to release the MediaPlayer after playback is completed.
mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
```

```
@Override
    public void onCompletion(MediaPlayer mp) {
        mp.release();
    }
});
```

Code examples for Amazon Polly using AWS SDKs

The following code examples show how to use Amazon Polly with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Code examples

- [Basic examples for Amazon Polly using AWS SDKs](#)
 - [Actions for Amazon Polly using AWS SDKs](#)
 - [Use DeleteLexicon with an AWS SDK or CLI](#)
 - [Use DescribeVoices with an AWS SDK](#)
 - [Use GetLexicon with an AWS SDK or CLI](#)
 - [Use GetSpeechSynthesisTask with an AWS SDK or CLI](#)
 - [Use ListLexicons with an AWS SDK or CLI](#)
 - [Use ListSpeechSynthesisTasks with an AWS SDK or CLI](#)
 - [Use PutLexicon with an AWS SDK or CLI](#)
 - [Use StartSpeechSynthesisTask with an AWS SDK or CLI](#)
 - [Use SynthesizeSpeech with an AWS SDK](#)
 - [Scenarios for Amazon Polly using AWS SDKs](#)
 - [Convert text to speech and back to text using an AWS SDK](#)
 - [Create a lip-sync application with Amazon Polly using an AWS SDK](#)
 - [Create an application that analyzes customer feedback and synthesizes audio](#)
 - [Getting started with text-to-speech synthesis](#)

Basic examples for Amazon Polly using AWS SDKs

The following code examples show how to use the basics of Amazon Polly with AWS SDKs.

Examples

- [Actions for Amazon Polly using AWS SDKs](#)
 - [Use DeleteLexicon with an AWS SDK or CLI](#)
 - [Use DescribeVoices with an AWS SDK](#)
 - [Use GetLexicon with an AWS SDK or CLI](#)
 - [Use GetSpeechSynthesisTask with an AWS SDK or CLI](#)
 - [Use ListLexicons with an AWS SDK or CLI](#)
 - [Use ListSpeechSynthesisTasks with an AWS SDK or CLI](#)
 - [Use PutLexicon with an AWS SDK or CLI](#)
 - [Use StartSpeechSynthesisTask with an AWS SDK or CLI](#)
 - [Use SynthesizeSpeech with an AWS SDK](#)

Actions for Amazon Polly using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Polly actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon Polly API and are code excerpts from larger programs that must be run in context. You can see actions in context in [Scenarios for Amazon Polly using AWS SDKs](#).

The following examples include only the most commonly used actions. For a complete list, see the [Amazon Polly API Reference](#).

Examples

- [Use DeleteLexicon with an AWS SDK or CLI](#)
- [Use DescribeVoices with an AWS SDK](#)
- [Use GetLexicon with an AWS SDK or CLI](#)
- [Use GetSpeechSynthesisTask with an AWS SDK or CLI](#)
- [Use ListLexicons with an AWS SDK or CLI](#)

- [Use ListSpeechSynthesisTasks with an AWS SDK or CLI](#)
- [Use PutLexicon with an AWS SDK or CLI](#)
- [Use StartSpeechSynthesisTask with an AWS SDK or CLI](#)
- [Use SynthesizeSpeech with an AWS SDK](#)

Use DeleteLexicon with an AWS SDK or CLI

The following code examples show how to use DeleteLexicon.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Getting started with text-to-speech synthesis](#)

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

/// <summary>
/// Deletes an existing Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class DeleteLexicon
{
    public static async Task Main()
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();
```

```

        var success = await DeletePollyLexiconAsync(client, lexiconName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {lexiconName}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {lexiconName}.");
        }
    }

    /// <summary>
    /// Deletes the named Amazon Polly lexicon.
    /// </summary>
    /// <param name="client">The initialized Amazon Polly client object.</
param>
    /// <param name="lexiconName">The name of the Amazon Polly lexicon to
    /// delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public static async Task<bool> DeletePollyLexiconAsync(
        AmazonPollyClient client,
        string lexiconName)
    {
        var deleteLexiconRequest = new DeleteLexiconRequest()
        {
            Name = lexiconName,
        };

        var response = await client.DeleteLexiconAsync(deleteLexiconRequest);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}

```

- For API details, see [DeleteLexicon](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To delete a lexicon

The following `delete-lexicon` example deletes the specified lexicon.

```
aws polly delete-lexicon \  
  --name w3c
```

This command produces no output.

For more information, see [Using the DeleteLexicon operation](#) in the *Amazon Polly Developer Guide*.

- For API details, see [DeleteLexicon](#) in *AWS CLI Command Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.  
  lo_ply->deletelexicon( iv_name ).  
  MESSAGE 'Lexicon deleted successfully.' TYPE 'I'.  
CATCH /aws1/cx_plylexiconnotfoundex.  
  MESSAGE 'Lexicon not found.' TYPE 'E'.  
CATCH /aws1/cx_plyservicefailureex.  
  MESSAGE 'Service failure occurred.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [DeleteLexicon](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use DescribeVoices with an AWS SDK

The following code examples show how to use DescribeVoices.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Getting started with text-to-speech synthesis](#)

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class DescribeVoices
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();

        var allVoicesRequest = new DescribeVoicesRequest();
        var enUsVoicesRequest = new DescribeVoicesRequest()
        {
            LanguageCode = "en-US",
        };

        try
```

```

        {
            string nextToken;
            do
            {
                var allVoicesResponse = await
client.DescribeVoicesAsync(allVoicesRequest);
                nextToken = allVoicesResponse.NextToken;
                allVoicesRequest.NextToken = nextToken;

                Console.WriteLine("\nAll voices: ");
                allVoicesResponse.Voices.ForEach(voice =>
                {
                    DisplayVoiceInfo(voice);
                });
            }
            while (nextToken is not null);

            do
            {
                var enUsVoicesResponse = await
client.DescribeVoicesAsync(enUsVoicesRequest);
                nextToken = enUsVoicesResponse.NextToken;
                enUsVoicesRequest.NextToken = nextToken;

                Console.WriteLine("\nen-US voices: ");
                enUsVoicesResponse.Voices.ForEach(voice =>
                {
                    DisplayVoiceInfo(voice);
                });
            }
            while (nextToken is not null);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Exception caught: " + ex.Message);
        }
    }

    public static void DisplayVoiceInfo(Voice voice)
    {
        Console.WriteLine($" Name: {voice.Name}\tGender:
{voice.Gender}\tLanguageName: {voice.LanguageName}");
    }
}

```

- For API details, see [DescribeVoices](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.DescribeVoicesRequest;
import software.amazon.awssdk.services.polly.model.DescribeVoicesResponse;
import software.amazon.awssdk.services.polly.model.PollyException;
import software.amazon.awssdk.services.polly.model.Voice;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class DescribeVoicesSample {
    public static void main(String args[]) {
        PollyClient polly = PollyClient.builder()
            .region(Region.US_WEST_2)
            .build();

        describeVoice(polly);
        polly.close();
    }
}
```

```

public static void describeVoice(PollyClient polly) {
    try {
        DescribeVoicesRequest voicesRequest = DescribeVoicesRequest.builder()
            .languageCode("en-US")
            .build();

        DescribeVoicesResponse enUsVoicesResult =
polly.describeVoices(voicesRequest);
        List<Voice> voices = enUsVoicesResult.voices();
        for (Voice myVoice : voices) {
            System.out.println("The ID of the voice is " + myVoice.id());
            System.out.println("The gender of the voice is " +
myVoice.gender());
        }

        } catch (PollyException e) {
            System.err.println("Exception caught: " + e);
            System.exit(1);
        }
    }
}

```

- For API details, see [DescribeVoices](#) in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

    def __init__(self, polly_client, s3_resource):
        """
        :param polly_client: A Boto3 Amazon Polly client.

```



```

        :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
resource.
        """
        self.polly_client = polly_client
        self.s3_resource = s3_resource
        self.voice_metadata = None

    def describe_voices(self):
        """
        Gets metadata about available voices.

        :return: The list of voice metadata.
        """
        try:
            response = self.polly_client.describe_voices()
            self.voice_metadata = response["Voices"]
            logger.info("Got metadata about %s voices.",
len(self.voice_metadata))
        except ClientError:
            logger.exception("Couldn't get voice metadata.")
            raise
        else:
            return self.voice_metadata

```

- For API details, see [DescribeVoices](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require 'aws-sdk-polly' # In v2: require 'aws-sdk'
```

```
begin
  # Create an Amazon Polly client using
  # credentials from the shared credentials file ~/.aws/credentials
  # and the configuration (region) from the shared configuration file ~/.aws/
  config
  polly = Aws::Polly::Client.new

  # Get US English voices
  resp = polly.describe_voices(language_code: 'en-US')

  resp.voices.each do |v|
    puts v.name
    puts "  #{v.gender}"
    puts
  end
rescue StandardError => e
  puts 'Could not get voices'
  puts 'Error message:'
  puts e.message
end
```

- For API details, see [DescribeVoices](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn list_voices(client: &Client) -> Result<(), Error> {
  let resp = client.describe_voices().send().await?;

  println!("Voices:");

  let voices = resp.voices();
  for voice in voices {
    println!("  Name:      {}", voice.name().unwrap_or("No name!"));
  }
}
```

```
println!(
    "  Language: {}",
    voice.language_name().unwrap_or("No language!")
);

println!();
}

println!("Found {} voices", voices.len());

Ok(())
}
```

- For API details, see [DescribeVoices](#) in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.
    " Only pass optional parameters if they have values
    IF iv_engine IS NOT INITIAL AND iv_language IS NOT INITIAL.
        oo_result = lo_ply->describevoices(
            iv_engine = iv_engine
            iv_languagecode = iv_language ).
    ELSEIF iv_engine IS NOT INITIAL.
        oo_result = lo_ply->describevoices(
            iv_engine = iv_engine ).
    ELSEIF iv_language IS NOT INITIAL.
        oo_result = lo_ply->describevoices(
            iv_languagecode = iv_language ).
    ELSE.
        oo_result = lo_ply->describevoices( ).
    ENDIF.
    MESSAGE 'Retrieved voice metadata.' TYPE 'I'.
```

```
CATCH /aws1/cx_plyinvalidnexttokenex.  
    MESSAGE 'The NextToken is invalid.' TYPE 'E'.  
CATCH /aws1/cx_plyservicefailureex.  
    MESSAGE 'Service failure occurred.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [DescribeVoices](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetLexicon with an AWS SDK or CLI

The following code examples show how to use GetLexicon.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Getting started with text-to-speech synthesis](#)

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;  
using System.Threading.Tasks;  
using Amazon.Polly;  
using Amazon.Polly.Model;  
  
/// <summary>  
/// Retrieves information about a specific Amazon Polly lexicon.  
/// </summary>
```

```
public class GetLexicon
{
    public static async Task Main(string[] args)
    {
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();

        await GetPollyLexiconAsync(client, lexiconName);
    }

    public static async Task GetPollyLexiconAsync(AmazonPollyClient client,
string lexiconName)
    {
        var getLexiconRequest = new GetLexiconRequest()
        {
            Name = lexiconName,
        };

        try
        {
            var response = await client.GetLexiconAsync(getLexiconRequest);
            Console.WriteLine($"Lexicon:\n Name: {response.Lexicon.Name}");
            Console.WriteLine($"Content: {response.Lexicon.Content}");
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error: " + ex.Message);
        }
    }
}
```

- For API details, see [GetLexicon](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To retrieve the content of a lexicon

The following `get-lexicon` example retrieves the content of the specified pronunciation lexicon.

```
aws polly get-lexicon \
  --name w3c
```

Output:

```
{
  "Lexicon": {
    "Content": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<lexicon version=
\"1.0\" \n      xmlns=      \"http://www.w3.org/2005/01/pronunciation-lexicon
\" \n      xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \n
xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-lexicon \n
http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" \n
  alphabet=\"ipa\" \n      xml:lang=\"en-US\">\n  <lexeme>\n    <grapheme>W3C</
grapheme>\n      <alias>World Wide Web Consortium</alias>\n  </lexeme>\n</
lexicon>\n",
    "Name": "w3c"
  },
  "LexiconAttributes": {
    "Alphabet": "ipa",
    "LanguageCode": "en-US",
    "LastModified": 1603908910.99,
    "LexiconArn": "arn:aws:polly:us-west-2:880185128111:lexicon/w3c",
    "LexemesCount": 1,
    "Size": 492
  }
}
```

For more information, see [Using the GetLexicon operation](#) in the *Amazon Polly Developer Guide*.

- For API details, see [GetLexicon](#) in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

    def __init__(self, polly_client, s3_resource):
        """
        :param polly_client: A Boto3 Amazon Polly client.
        :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
        resource.
        """
        self.polly_client = polly_client
        self.s3_resource = s3_resource
        self.voice_metadata = None

    def get_lexicon(self, name):
        """
        Gets metadata and contents of an existing lexicon.

        :param name: The name of the lexicon to retrieve.
        :return: The retrieved lexicon.
        """
        try:
            response = self.polly_client.get_lexicon(Name=name)
            logger.info("Got lexicon %s.", name)
        except ClientError:
            logger.exception("Couldn't get lexicon %s.", name)
            raise
        else:
            return response
```

- For API details, see [GetLexicon](#) in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.  
    oo_result = lo_ply->getlexicon( iv_name ).  
    DATA(lo_lexicon) = oo_result->get_lexicon( ).  
    IF lo_lexicon IS BOUND.  
        DATA(lv_lex_name) = lo_lexicon->get_name( ).  
        MESSAGE |Retrieved lexicon: { lv_lex_name }| TYPE 'I'.  
    ENDIF.  
    CATCH /aws1/cx_plylexiconnotfoundex.  
        MESSAGE 'Lexicon not found.' TYPE 'E'.  
    CATCH /aws1/cx_plyservicefailureex.  
        MESSAGE 'Service failure occurred.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [GetLexicon](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use GetSpeechSynthesisTask with an AWS SDK or CLI

The following code examples show how to use GetSpeechSynthesisTask.

CLI

AWS CLI

To get information about a speech synthesis task

The following `get-speech-synthesis-task` example retrieves information about the specified speech synthesis task.

```
aws polly get-speech-synthesis-task \
  --task-id 70b61c0f-57ce-4715-a247-cae8729dcce9
```

Output:

```
{
  "SynthesisTask": {
    "TaskId": "70b61c0f-57ce-4715-a247-cae8729dcce9",
    "TaskStatus": "completed",
    "OutputUri": "https://s3.us-west-2.amazonaws.com/amzn-s3-demo-
bucket/70b61c0f-57ce-4715-a247-cae8729dcce9.mp3",
    "CreationTime": 1603911042.689,
    "RequestCharacters": 1311,
    "OutputFormat": "mp3",
    "TextType": "text",
    "VoiceId": "Joanna"
  }
}
```

For more information, see [Creating long audio files](#) in the *Amazon Polly Developer Guide*.

- For API details, see [GetSpeechSynthesisTask](#) in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

 Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

    def __init__(self, polly_client, s3_resource):
        """
        :param polly_client: A Boto3 Amazon Polly client.
        :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
        resource.
        """
        self.polly_client = polly_client
        self.s3_resource = s3_resource
        self.voice_metadata = None

    def get_speech_synthesis_task(self, task_id):
        """
        Gets metadata about an asynchronous speech synthesis task, such as its
        status.

        :param task_id: The ID of the task to retrieve.
        :return: Metadata about the task.
        """
        try:
            response =
self.polly_client.get_speech_synthesis_task(TaskId=task_id)
            task = response["SynthesisTask"]
            logger.info("Got synthesis task. Status is %s.", task["TaskStatus"])
        except ClientError:
            logger.exception("Couldn't get synthesis task %s.", task_id)
            raise
        else:
            return task

```

- For API details, see [GetSpeechSynthesisTask](#) in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.  
    oo_result = lo_ply->getspeechsynthesistask( iv_task_id ).  
    DATA(lo_task) = oo_result->get_synthesistask( ).  
    IF lo_task IS BOUND.  
        DATA(lv_status) = lo_task->get_taskstatus( ).  
        MESSAGE |Task status: { lv_status }| TYPE 'I'.  
    ENDIF.  
    CATCH /aws1/cx_plyinvalidtaskidex.  
        MESSAGE 'Invalid task ID.' TYPE 'E'.  
    CATCH /aws1/cx_plyservicefailureex.  
        MESSAGE 'Service failure occurred.' TYPE 'E'.  
    CATCH /aws1/cx_plysynthesistsknotf00.  
        MESSAGE 'Synthesis task not found.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [GetSpeechSynthesisTask](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ListLexicons with an AWS SDK or CLI

The following code examples show how to use ListLexicons.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Getting started with text-to-speech synthesis](#)

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

/// <summary>
/// Lists the Amazon Polly lexicons that have been defined. By default,
/// lists the lexicons that are defined in the same AWS Region as the default
/// user. To view Amazon Polly lexicons that are defined in a different AWS
/// Region, supply it as a parameter to the Amazon Polly constructor.
/// </summary>
public class ListLexicons
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        var request = new ListLexiconsRequest();

        try
        {
            Console.WriteLine("All voices: ");

            do
            {
                var response = await client.ListLexiconsAsync(request);
                request.NextToken = response.NextToken;

                response.Lexicons.ForEach(lexicon =>
                {
                    var attributes = lexicon.Attributes;
                    Console.WriteLine($"Name: {lexicon.Name}");
                    Console.WriteLine($"\\tAlphabet: {attributes.Alphabet}");
                });
            } while (request.NextToken != null);
        }
    }
}
```

```

        Console.WriteLine($"\\tLanguageCode:
{attributes.LanguageCode}");
        Console.WriteLine($"\\tLastModified:
{attributes.LastModified}");
        Console.WriteLine($"\\tLexemesCount:
{attributes.LexemesCount}");
        Console.WriteLine($"\\tLexiconArn:
{attributes.LexiconArn}");
        Console.WriteLine($"\\tSize: {attributes.Size}");
    });
}
while (request.NextToken is not null);
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
}
}

```

- For API details, see [ListLexicons](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list your lexicons

The following `list-lexicons` example lists your pronunciation lexicons.

```
aws polly list-lexicons
```

Output:

```

{
  "Lexicons": [
    {
      "Name": "w3c",
      "Attributes": {
        "Alphabet": "ipa",
        "LanguageCode": "en-US",

```

```

        "LastModified": 1603908910.99,
        "LexiconArn": "arn:aws:polly:us-east-2:123456789012:lexicon/w3c",
        "LexemesCount": 1,
        "Size": 492
    }
}
]
}

```

For more information, see [Using the ListLexicons operation](#) in the *Amazon Polly Developer Guide*.

- For API details, see [ListLexicons](#) in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.ListLexiconsResponse;
import software.amazon.awssdk.services.polly.model.ListLexiconsRequest;
import software.amazon.awssdk.services.polly.model.LexiconDescription;
import software.amazon.awssdk.services.polly.model.PollyException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class ListLexicons {

```

```
public static void main(String args[]) {
    PollyClient polly = PollyClient.builder()
        .region(Region.US_WEST_2)
        .build();

    listLexicons(polly);
    polly.close();
}

public static void listLexicons(PollyClient client) {
    try {
        ListLexiconsRequest listLexiconsRequest =
ListLexiconsRequest.builder()
            .build();

        ListLexiconsResponse listLexiconsResult =
client.listLexicons(listLexiconsRequest);
        List<LexiconDescription> lexiconDescription =
listLexiconsResult.lexicons();
        for (LexiconDescription lexDescription : lexiconDescription) {
            System.out.println("The name of the Lexicon is " +
lexDescription.name());
        }

    } catch (PollyException e) {
        System.err.println("Exception caught: " + e);
        System.exit(1);
    }
}
```

- For API details, see [ListLexicons](#) in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

    def __init__(self, polly_client, s3_resource):
        """
        :param polly_client: A Boto3 Amazon Polly client.
        :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
        resource.
        """
        self.polly_client = polly_client
        self.s3_resource = s3_resource
        self.voice_metadata = None

    def list_lexicons(self):
        """
        Lists lexicons in the current account.

        :return: The list of lexicons.
        """
        try:
            response = self.polly_client.list_lexicons()
            lexicons = response["Lexicons"]
            logger.info("Got %s lexicons.", len(lexicons))
        except ClientError:
            logger.exception(
                "Couldn't get %s.",
            )
            raise
        else:
            return lexicons
```

- For API details, see [ListLexicons](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
require 'aws-sdk-polly' # In v2: require 'aws-sdk'

begin
  # Create an Amazon Polly client using
  # credentials from the shared credentials file ~/.aws/credentials
  # and the configuration (region) from the shared configuration file ~/.aws/
  config
  polly = Aws::Polly::Client.new

  resp = polly.list_lexicons

  resp.lexicons.each do |l|
    puts l.name
    puts "  Alphabet:#{l.attributes.alphabet}"
    puts "  Language:#{l.attributes.language}"
    puts
  end
rescue StandardError => e
  puts 'Could not get lexicons'
  puts 'Error message:'
  puts e.message
end
```

- For API details, see [ListLexicons](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn show_lexicons(client: &Client) -> Result<(), Error> {
    let resp = client.list_lexicons().send().await?;

    println!("Lexicons:");

    let lexicons = resp.lexicons();

    for lexicon in lexicons {
        println!("  Name:      {}", lexicon.name().unwrap_or_default());
        println!(
            "    Language: {:?}\n",
            lexicon
                .attributes()
                .as_ref()
                .map(|attrib| attrib
                    .language_code
                    .as_ref()
                    .expect("languages must have language codes"))
                .expect("languages must have attributes")
        );
    }

    println();
    println!("Found {} lexicons.", lexicons.len());
    println();

    Ok(())
}
```

- For API details, see [ListLexicons](#) in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.  
    oo_result = lo_ply->listlexicons( ).  
    DATA(lt_lexicons) = oo_result->get_lexicons( ).  
    DATA(lv_count) = lines( lt_lexicons ).  
    MESSAGE |Found { lv_count } lexicons| TYPE 'I'.  
CATCH /aws1/cx_plyinvalidnexttokenex.  
    MESSAGE 'Invalid NextToken.' TYPE 'E'.  
CATCH /aws1/cx_plyservicefailureex.  
    MESSAGE 'Service failure occurred.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [ListLexicons](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use ListSpeechSynthesisTasks with an AWS SDK or CLI

The following code examples show how to use ListSpeechSynthesisTasks.

CLI

AWS CLI

To list your speech synthesis tasks

The following `list-speech-synthesis-tasks` example lists your speech synthesis tasks.

```
aws polly list-speech-synthesis-tasks
```

Output:

```
{
  "SynthesisTasks": [
    {
      "TaskId": "70b61c0f-57ce-4715-a247-cae8729dcce9",
      "TaskStatus": "completed",
      "OutputUri": "https://s3.us-west-2.amazonaws.com/amzn-s3-demo-
bucket/70b61c0f-57ce-4715-a247-cae8729dcce9.mp3",
      "CreationTime": 1603911042.689,
      "RequestCharacters": 1311,
      "OutputFormat": "mp3",
      "TextType": "text",
      "VoiceId": "Joanna"
    }
  ]
}
```

For more information, see [Creating long audio files](#) in the *Amazon Polly Developer Guide*.

- For API details, see [ListSpeechSynthesisTasks](#) in *AWS CLI Command Reference*.

SAP ABAP**SDK for SAP ABAP****Note**

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

TRY.

```
" Only pass optional parameters if they have values
IF iv_max_results IS NOT INITIAL AND iv_status IS NOT INITIAL.
  oo_result = lo_ply->listspeechsynthesistasks(
    iv_maxresults = iv_max_results
    iv_status = iv_status ).
ELSEIF iv_max_results IS NOT INITIAL.
  oo_result = lo_ply->listspeechsynthesistasks(
    iv_maxresults = iv_max_results ).
```

```

ELSEIF iv_status IS NOT INITIAL.
    oo_result = lo_ply->listspeechsynthesistasks(
        iv_status = iv_status ).
ELSE.
    oo_result = lo_ply->listspeechsynthesistasks( ).
ENDIF.
DATA(lt_tasks) = oo_result->get_synthesistasks( ).
DATA(lv_count) = lines( lt_tasks ).
MESSAGE |Found { lv_count } synthesis tasks| TYPE 'I'.
CATCH /aws1/cx_plyinvalidnexttokenex.
    MESSAGE 'Invalid NextToken.' TYPE 'E'.
CATCH /aws1/cx_plyservicefailureex.
    MESSAGE 'Service failure occurred.' TYPE 'E'.
ENDTRY.

```

- For API details, see [ListSpeechSynthesisTasks](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use PutLexicon with an AWS SDK or CLI

The following code examples show how to use PutLexicon.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Getting started with text-to-speech synthesis](#)

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

using System;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

/// <summary>
/// Creates a new Amazon Polly lexicon using the AWS SDK for .NET.
/// </summary>
public class PutLexicon
{
    public static async Task Main()
    {
        string lexiconContent = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
+
        "<lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/
pronunciation-lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" "
+
        "xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-
lexicon http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\" " +
        "alphabet=\"ipa\" xml:lang=\"en-US\">" +
        "<lexeme><grapheme>test1</grapheme><alias>test2</alias></lexeme>"
+
        "</lexicon>";
        string lexiconName = "SampleLexicon";

        var client = new AmazonPollyClient();
        var putLexiconRequest = new PutLexiconRequest()
        {
            Name = lexiconName,
            Content = lexiconContent,
        };

        try
        {
            var response = await client.PutLexiconAsync(putLexiconRequest);
            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"Successfully created Lexicon:
{lexiconName}.");
            }
            else
            {

```

```

        Console.WriteLine($"Could not create Lexicon:
{lexiconName}.");
    }
}
catch (Exception ex)
{
    Console.WriteLine("Exception caught: " + ex.Message);
}
}
}

```

- For API details, see [PutLexicon](#) in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To store a lexicon

The following `put-lexicon` example stores the specified pronunciation lexicon. The `example.pls` file specifies a W3C PLS-compliant lexicon.

```

aws polly put-lexicon \
  --name w3c \
  --content file://example.pls

```

Contents of `example.pls`

```

{
  <?xml version="1.0" encoding="UTF-8"?>
  <lexicon version="1.0"
    xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
      http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
    alphabet="ipa"
    xml:lang="en-US">
    <lexeme>
      <grapheme>W3C</grapheme>
      <alias>World Wide Web Consortium</alias>
    </lexeme>
  </lexicon>
}

```

```

    </lexeme>
  </lexicon>
}

```

This command produces no output.

For more information, see [Using the PutLexicon operation](#) in the *Amazon Polly Developer Guide*.

- For API details, see [PutLexicon](#) in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

    def __init__(self, polly_client, s3_resource):
        """
        :param polly_client: A Boto3 Amazon Polly client.
        :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
        resource.
        """
        self.polly_client = polly_client
        self.s3_resource = s3_resource
        self.voice_metadata = None

    def create_lexicon(self, name, content):
        """
        Creates a lexicon with the specified content. A lexicon contains custom
        pronunciations.

        :param name: The name of the lexicon.
        :param content: The content of the lexicon.

```



```

"""
try:
    self.polly_client.put_lexicon(Name=name, Content=content)
    logger.info("Created lexicon %s.", name)
except ClientError:
    logger.exception("Couldn't create lexicon %s.")
    raise

```

- For API details, see [PutLexicon](#) in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

async fn make_lexicon(client: &Client, name: &str, from: &str, to: &str) ->
Result<(), Error> {
    let content = format!("<?xml version=\"1.0\" encoding=\"UTF-8\"?>
    <lexicon version=\"1.0\" xmlns=\"http://www.w3.org/2005/01/pronunciation-
lexicon\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
    xsi:schemaLocation=\"http://www.w3.org/2005/01/pronunciation-lexicon http://
www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd\"
    alphabet=\"ipa\" xml:lang=\"en-US\">
    <lexeme><grapheme>{}</grapheme><alias>{}</alias></lexeme>
    </lexicon>", from, to);

    client
        .put_lexicon()
        .name(name)
        .content(content)
        .send()
        .await?;

    println!("Added lexicon");

```

```
    Ok(() )
}
```

- For API details, see [PutLexicon](#) in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.
    lo_ply->putlexicon(
        iv_name = iv_name
        iv_content = iv_content ).
    MESSAGE 'Lexicon created successfully.' TYPE 'I'.
CATCH /aws1/cx_plyinvalidlexiconex.
    MESSAGE 'Invalid lexicon.' TYPE 'E'.
CATCH /aws1/cx_plylexiconsizeexcdex.
    MESSAGE 'Lexicon size exceeded.' TYPE 'E'.
CATCH /aws1/cx_plymaxlexemelengthe00.
    MESSAGE 'Maximum lexeme length exceeded.' TYPE 'E'.
CATCH /aws1/cx_plymaxlexiconsnoexc00.
    MESSAGE 'Maximum number of lexicons exceeded.' TYPE 'E'.
CATCH /aws1/cx_plyservicefailureex.
    MESSAGE 'Service failure occurred.' TYPE 'E'.
CATCH /aws1/cx_plyunsuppdpalsalpha00.
    MESSAGE 'Unsupported PLS alphabet.' TYPE 'E'.
CATCH /aws1/cx_plyunsuppdpplslangu00.
    MESSAGE 'Unsupported PLS language.' TYPE 'E'.
ENDTRY.
```

- For API details, see [PutLexicon](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use StartSpeechSynthesisTask with an AWS SDK or CLI

The following code examples show how to use StartSpeechSynthesisTask.

CLI

AWS CLI

To synthesize text

The following start-speech-synthesis-task example synthesizes the text in `text_file.txt` and stores the resulting MP3 file in the specified bucket.

```
aws polly start-speech-synthesis-task \
  --output-format mp3 \
  --output-s3-bucket-name amzn-s3-demo-bucket \
  --text file://text_file.txt \
  --voice-id Joanna
```

Output:

```
{
  "SynthesisTask": {
    "TaskId": "70b61c0f-57ce-4715-a247-cae8729dcce9",
    "TaskStatus": "scheduled",
    "OutputUri": "https://s3.us-east-2.amazonaws.com/amzn-s3-demo-bucket/70b61c0f-57ce-4715-a247-cae8729dcce9.mp3",
    "CreationTime": 1603911042.689,
    "RequestCharacters": 1311,
    "OutputFormat": "mp3",
    "TextType": "text",
    "VoiceId": "Joanna"
  }
}
```

For more information, see [Creating long audio files](#) in the *Amazon Polly Developer Guide*.

- For API details, see [StartSpeechSynthesisTask](#) in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

 Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

    def __init__(self, polly_client, s3_resource):
        """
        :param polly_client: A Boto3 Amazon Polly client.
        :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
        resource.
        """
        self.polly_client = polly_client
        self.s3_resource = s3_resource
        self.voice_metadata = None

    def do_synthesis_task(
        self,
        text,
        engine,
        voice,
        audio_format,
        s3_bucket,
        lang_code=None,
        include_visemes=False,
        wait_callback=None,
    ):
        """
        Start an asynchronous task to synthesize speech or speech marks, wait for
        the task to complete, retrieve the output from Amazon S3, and return the
        data.

        An asynchronous task is required when the text is too long for near-real
        time

```

```

        synthesis.

        :param text: The text to synthesize.
        :param engine: The kind of engine used. Can be standard or neural.
        :param voice: The ID of the voice to use.
        :param audio_format: The audio format to return for synthesized speech.
When
        speech marks are synthesized, the output format is
JSON.
        :param s3_bucket: The name of an existing Amazon S3 bucket that you have
        write access to. Synthesis output is written to this
bucket.
        :param lang_code: The language code of the voice to use. This has an
effect
        only when a bilingual voice is selected.
        :param include_visemes: When True, a second request is made to Amazon
Polly
        to synthesize a list of visemes, using the
specified
        text and voice. A viseme represents the visual
position
        of the face and mouth when saying part of a word.
        :param wait_callback: A callback function that is called periodically
during
        task processing, to give the caller an opportunity
to
        take action, such as to display status.
        :return: The audio stream that contains the synthesized speech and a list
        of visemes that are associated with the speech audio.
"""
try:
    kwargs = {
        "Engine": engine,
        "OutputFormat": audio_format,
        "OutputS3BucketName": s3_bucket,
        "Text": text,
        "VoiceId": voice,
    }
    if lang_code is not None:
        kwargs["LanguageCode"] = lang_code
    response = self.polly_client.start_speech_synthesis_task(**kwargs)
    speech_task = response["SynthesisTask"]
    logger.info("Started speech synthesis task %s.",
speech_task["TaskId"])

```

```

        viseme_task = None
        if include_visemes:
            kwargs["OutputFormat"] = "json"
            kwargs["SpeechMarkTypes"] = ["viseme"]
            response =
self.polly_client.start_speech_synthesis_task(**kwargs)
            viseme_task = response["SynthesisTask"]
            logger.info("Started viseme synthesis task %s.",
viseme_task["TaskId"])
        except ClientError:
            logger.exception("Couldn't start synthesis task.")
            raise
        else:
            bucket = self.s3_resource.Bucket(s3_bucket)
            audio_stream = self._wait_for_task(
                10, speech_task["TaskId"], "speech", wait_callback, bucket
            )

            visemes = None
            if include_visemes:
                viseme_data = self._wait_for_task(
                    10, viseme_task["TaskId"], "viseme", wait_callback, bucket
                )
                visemes = [
                    json.loads(v) for v in viseme_data.read().decode().split() if
v
                ]

            return audio_stream, visemes

```

- For API details, see [StartSpeechSynthesisTask](#) in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

 Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
TRY.
    " Only pass optional parameters if they have values
    IF iv_lang_code IS NOT INITIAL AND iv_s3_key_prefix IS NOT INITIAL.
        oo_result = lo_ply->startspeechsynthesistask(
            iv_engine = iv_engine
            iv_outputformat = iv_audio_format
            iv_outputs3bucketname = iv_s3_bucket
            iv_outputs3keyprefix = iv_s3_key_prefix
            iv_text = iv_text
            iv_voiceid = iv_voice_id
            iv_languagecode = iv_lang_code ).
    ELSEIF iv_lang_code IS NOT INITIAL.
        oo_result = lo_ply->startspeechsynthesistask(
            iv_engine = iv_engine
            iv_outputformat = iv_audio_format
            iv_outputs3bucketname = iv_s3_bucket
            iv_text = iv_text
            iv_voiceid = iv_voice_id
            iv_languagecode = iv_lang_code ).
    ELSEIF iv_s3_key_prefix IS NOT INITIAL.
        oo_result = lo_ply->startspeechsynthesistask(
            iv_engine = iv_engine
            iv_outputformat = iv_audio_format
            iv_outputs3bucketname = iv_s3_bucket
            iv_outputs3keyprefix = iv_s3_key_prefix
            iv_text = iv_text
            iv_voiceid = iv_voice_id ).
    ELSE.
        oo_result = lo_ply->startspeechsynthesistask(
            iv_engine = iv_engine
            iv_outputformat = iv_audio_format
            iv_outputs3bucketname = iv_s3_bucket
```

```

        iv_text = iv_text
        iv_voiceid = iv_voice_id ).
    ENDIF.
    MESSAGE 'Speech synthesis task started.' TYPE 'I'.
CATCH /aws1/cx_plyinvalids3bucketex.
    MESSAGE 'Invalid S3 bucket.' TYPE 'E'.
CATCH /aws1/cx_plyinvalidssmlex.
    MESSAGE 'Invalid SSML.' TYPE 'E'.
CATCH /aws1/cx_plylexiconnotfoundex.
    MESSAGE 'Lexicon not found.' TYPE 'E'.
CATCH /aws1/cx_plyservicefailureex.
    MESSAGE 'Service failure occurred.' TYPE 'E'.
CATCH /aws1/cx_plytextlengthexcdex.
    MESSAGE 'Text length exceeded maximum.' TYPE 'E'.
ENDTRY.

```

- For API details, see [StartSpeechSynthesisTask](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Use SynthesizeSpeech with an AWS SDK

The following code examples show how to use SynthesizeSpeech.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- [Getting started with text-to-speech synthesis](#)

.NET

SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).


```

using System;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

public class SynthesizeSpeech
{
    public static async Task Main()
    {
        string outputFileName = "speech.mp3";
        string text = "Twas brillig, and the slithy toves did gyre and gimbol
in the wabe";

        var client = new AmazonPollyClient();
        var response = await PollySynthesizeSpeech(client, text);

        WriteSpeechToStream(response.AudioStream, outputFileName);
    }

    /// <summary>
    /// Calls the Amazon Polly SynthesizeSpeechAsync method to convert text
    /// to speech.
    /// </summary>
    /// <param name="client">The Amazon Polly client object used to connect
    /// to the Amazon Polly service.</param>
    /// <param name="text">The text to convert to speech.</param>
    /// <returns>A SynthesizeSpeechResponse object that includes an
AudioStream
    /// object with the converted text.</returns>
    private static async Task<SynthesizeSpeechResponse>
PollySynthesizeSpeech(IAmazonPolly client, string text)
    {
        var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
        {
            OutputFormat = OutputFormat.Mp3,
            VoiceId = VoiceId.Joanna,
            Text = text,
        };

        var synthesizeSpeechResponse =
            await client.SynthesizeSpeechAsync(synthesizeSpeechRequest);
    }
}

```

```

        return synthesizeSpeechResponse;
    }

    /// <summary>
    /// Writes the AudioStream returned from the call to
    /// SynthesizeSpeechAsync to a file in MP3 format.
    /// </summary>
    /// <param name="audioStream">The AudioStream returned from the
    /// call to the SynthesizeSpeechAsync method.</param>
    /// <param name="outputFileName">The full path to the file in which to
    /// save the audio stream.</param>
    private static void WriteSpeechToStream(Stream audioStream, string
outputFileName)
    {
        var outputStream = new FileStream(
            outputFileName,
            FileMode.Create,
            FileAccess.Write);
        byte[] buffer = new byte[2 * 1024];
        int readBytes;

        while ((readBytes = audioStream.Read(buffer, 0, 2 * 1024)) > 0)
        {
            outputStream.Write(buffer, 0, readBytes);
        }

        // Flushes the buffer to avoid losing the last second or so of
        // the synthesized text.
        outputStream.Flush();
        Console.WriteLine($"Saved {outputFileName} to disk.");
    }
}

```

Synthesize speech from text using speech marks with Amazon Polly using an AWS SDK.

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Polly;
using Amazon.Polly.Model;

```

```

public class SynthesizeSpeechMarks
{
    public static async Task Main()
    {
        var client = new AmazonPollyClient();
        string outputFileName = "speechMarks.json";

        var synthesizeSpeechRequest = new SynthesizeSpeechRequest()
        {
            OutputFormat = OutputFormat.Json,
            SpeechMarkTypes = new List<string>
            {
                SpeechMarkType.Viseme,
                SpeechMarkType.Word,
            },
            VoiceId = VoiceId.Joanna,
            Text = "This is a sample text to be synthesized.",
        };

        try
        {
            using (var outputStream = new FileStream(outputFileName,
                FileMode.Create, FileAccess.Write))
            {
                var synthesizeSpeechResponse = await
client.SynthesizeSpeechAsync(synthesizeSpeechRequest);
                var buffer = new byte[2 * 1024];
                int readBytes;

                var inputStream = synthesizeSpeechResponse.AudioStream;
                while ((readBytes = inputStream.Read(buffer, 0, 2 * 1024)) >
0)
                {
                    outputStream.Write(buffer, 0, readBytes);
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }
}

```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
import javazoom.jl.decoder.JavaLayerException;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.polly.PollyClient;
import software.amazon.awssdk.services.polly.model.DescribeVoicesRequest;
import software.amazon.awssdk.services.polly.model.Voice;
import software.amazon.awssdk.services.polly.model.DescribeVoicesResponse;
import software.amazon.awssdk.services.polly.model.OutputFormat;
import software.amazon.awssdk.services.polly.model.PollyException;
import software.amazon.awssdk.services.polly.model.SynthesizeSpeechRequest;
import software.amazon.awssdk.services.polly.model.SynthesizeSpeechResponse;
import java.io.IOException;
import java.io.InputStream;
import javazoom.jl.player.advanced.AdvancedPlayer;
import javazoom.jl.player.advanced.PlaybackEvent;
import javazoom.jl.player.advanced.PlaybackListener;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class PollyDemo {
```

```
private static final String SAMPLE = "Congratulations. You have successfully
built this working demo " +
    " of Amazon Polly in Java Version 2. Have fun building voice enabled
apps with Amazon Polly (that's me!), and always "
    +
    " look at the AWS website for tips and tricks on using Amazon Polly
and other great services from AWS";

public static void main(String args[]) {
    PollyClient polly = PollyClient.builder()
        .region(Region.US_WEST_2)
        .build();

    talkPolly(polly);
    polly.close();
}

public static void talkPolly(PollyClient polly) {
    try {
        DescribeVoicesRequest describeVoiceRequest =
DescribeVoicesRequest.builder()
            .engine("standard")
            .build();

        DescribeVoicesResponse describeVoicesResult =
polly.describeVoices(describeVoiceRequest);
        Voice voice = describeVoicesResult.voices().stream()
            .filter(v -> v.name().equals("Joanna"))
            .findFirst()
            .orElseThrow(() -> new RuntimeException("Voice not found"));
        InputStream stream = synthesize(polly, SAMPLE, voice,
OutputFormat.MP3);
        AdvancedPlayer player = new AdvancedPlayer(stream,
javazoom.jl.player.FactoryRegistry.systemRegistry().createAudioDevice());
        player.setPlayBackListener(new PlaybackListener() {
            public void playbackStarted(PlaybackEvent evt) {
                System.out.println("Playback started");
                System.out.println(SAMPLE);
            }

            public void playbackFinished(PlaybackEvent evt) {
                System.out.println("Playback finished");
            }
        })
    }
}
```

```

        });

        // play it!
        player.play();

    } catch (PollyException | JavaLayerException | IOException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static InputStream synthesize(PollyClient polly, String text, Voice
voice, OutputFormat format)
    throws IOException {
    SynthesizeSpeechRequest synthReq = SynthesizeSpeechRequest.builder()
        .text(text)
        .voiceId(voice.id())
        .outputFormat(format)
        .build();

    ResponseInputStream<SynthesizeSpeechResponse> synthRes =
polly.synthesizeSpeech(synthReq);
    return synthRes;
}
}

```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

class PollyWrapper:
    """Encapsulates Amazon Polly functions."""

```

```

def __init__(self, polly_client, s3_resource):
    """
    :param polly_client: A Boto3 Amazon Polly client.
    :param s3_resource: A Boto3 Amazon Simple Storage Service (Amazon S3)
resource.
    """
    self.polly_client = polly_client
    self.s3_resource = s3_resource
    self.voice_metadata = None

    def synthesize(
        self, text, engine, voice, audio_format, lang_code=None,
include_visemes=False
    ):
        """
        Synthesizes speech or speech marks from text, using the specified voice.

        :param text: The text to synthesize.
        :param engine: The kind of engine used. Can be standard or neural.
        :param voice: The ID of the voice to use.
        :param audio_format: The audio format to return for synthesized speech.
When
                                speech marks are synthesized, the output format is
JSON.
        :param lang_code: The language code of the voice to use. This has an
effect
                                only when a bilingual voice is selected.
        :param include_visemes: When True, a second request is made to Amazon
Polly
                                to synthesize a list of visemes, using the
specified
                                text and voice. A viseme represents the visual
position
                                of the face and mouth when saying part of a word.
        :return: The audio stream that contains the synthesized speech and a list
of visemes that are associated with the speech audio.
        """
        try:
            kwargs = {
                "Engine": engine,
                "OutputFormat": audio_format,
                "Text": text,
                "VoiceId": voice,

```

```

    }
    if lang_code is not None:
        kwargs["LanguageCode"] = lang_code
    response = self.polly_client.synthesize_speech(**kwargs)
    audio_stream = response["AudioStream"]
    logger.info("Got audio stream spoken by %s.", voice)
    visemes = None
    if include_visemes:
        kwargs["OutputFormat"] = "json"
        kwargs["SpeechMarkTypes"] = ["viseme"]
        response = self.polly_client.synthesize_speech(**kwargs)
        visemes = [
            json.loads(v)
            for v in response["AudioStream"].read().decode().split()
            if v
        ]
        logger.info("Got %s visemes.", len(visemes))
except ClientError:
    logger.exception("Couldn't get audio stream.")
    raise
else:
    return audio_stream, visemes

```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

require 'aws-sdk-polly' # In v2: require 'aws-sdk'

begin
  # Get the filename from the command line

```



```
if ARGV.empty?
  puts 'You must supply a filename'
  exit 1
end

filename = ARGV[0]

# Open file and get the contents as a string
if File.exist?(filename)
  contents = IO.read(filename)
else
  puts "No such file: #{filename}"
  exit 1
end

# Create an Amazon Polly client using
# credentials from the shared credentials file ~/.aws/credentials
# and the configuration (region) from the shared configuration file ~/.aws/
config
polly = Aws::Polly::Client.new

resp = polly.synthesize_speech({
                                output_format: 'mp3',
                                text: contents,
                                voice_id: 'Joanna'
                              })

# Save output
# Get just the file name
# abc/xyz.txt -> xyx.txt
name = File.basename(filename)

# Split up name so we get just the xyz part
parts = name.split('.')
first_part = parts[0]
mp3_file = "#{first_part}.mp3"

IO.copy_stream(resp.audio_stream, mp3_file)

puts "Wrote MP3 content to: #{mp3_file}"
rescue StandardError => e
  puts 'Got error:'
  puts 'Error message:'
  puts e.message
```

```
end
```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
async fn synthesize(client: &Client, filename: &str) -> Result<(), Error> {
    let content = fs::read_to_string(filename);

    let resp = client
        .synthesize_speech()
        .output_format(OutputFormat::Mp3)
        .text(content.unwrap())
        .voice_id(VoiceId::Joanna)
        .send()
        .await?;

    // Get MP3 data from response and save it
    let mut blob = resp
        .audio_stream
        .collect()
        .await
        .expect("failed to read data");

    let parts: Vec<&str> = filename.split('.').collect();
    let out_file = format!("{}", String::from(parts[0]), ".mp3");

    let mut file = tokio::fs::File::create(out_file)
        .await
        .expect("failed to create file");

    file.write_all_buf(&mut blob)
        .await
```

```

        .expect("failed to write to file");

    Ok(())
}

```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```

TRY.
    " Only pass optional language code if it has a value
    IF iv_lang_code IS NOT INITIAL.
        oo_result = lo_ply->synthesizespeech(
            iv_engine = iv_engine
            iv_outputformat = iv_output_fmt
            iv_text = iv_text
            iv_voiceid = iv_voice_id
            iv_languagecode = iv_lang_code ).
    ELSE.
        oo_result = lo_ply->synthesizespeech(
            iv_engine = iv_engine
            iv_outputformat = iv_output_fmt
            iv_text = iv_text
            iv_voiceid = iv_voice_id ).
    ENDIF.
    MESSAGE 'Speech synthesized successfully.' TYPE 'I'.
CATCH /aws1/cx_plyinvalidssmlex.
    MESSAGE 'Invalid SSML.' TYPE 'E'.
CATCH /aws1/cx_plylexiconnotfoundex.
    MESSAGE 'Lexicon not found.' TYPE 'E'.
CATCH /aws1/cx_plyservicefailureex.
    MESSAGE 'Service failure occurred.' TYPE 'E'.
CATCH /aws1/cx_plytextlengthexcdex.

```

```
MESSAGE 'Text length exceeded maximum.' TYPE 'E'.  
ENDTRY.
```

- For API details, see [SynthesizeSpeech](#) in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Polly using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Polly with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon Polly or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- [Convert text to speech and back to text using an AWS SDK](#)
- [Create a lip-sync application with Amazon Polly using an AWS SDK](#)
- [Create an application that analyzes customer feedback and synthesizes audio](#)
- [Getting started with text-to-speech synthesis](#)

Convert text to speech and back to text using an AWS SDK

The following code example shows how to:

- Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file.
- Upload the audio file to an Amazon S3 bucket.
- Use Amazon Transcribe to convert the audio file to text.
- Display the text.

Rust

SDK for Rust

Use Amazon Polly to synthesize a plain text (UTF-8) input file to an audio file, upload the audio file to an Amazon S3 bucket, use Amazon Transcribe to convert that audio file to text, and display the text.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Polly
- Amazon S3
- Amazon Transcribe

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Create a lip-sync application with Amazon Polly using an AWS SDK

The following code example shows how to create a lip-sync application with Amazon Polly.

Python

SDK for Python (Boto3)

Shows how to use Amazon Polly and Tkinter to create a lip-sync application that displays an animated face speaking along with the speech synthesized by Amazon Polly. Lip-sync is accomplished by requesting a list of visemes from Amazon Polly that match up with the synthesized speech.

- Get voice metadata from Amazon Polly and display it in a Tkinter application.
- Get synthesized speech audio and matching viseme speech marks from Amazon Polly.
- Play the audio with synchronized mouth movements in an animated face.
- Submit asynchronous synthesis tasks for long texts and retrieve the output from an Amazon Simple Storage Service (Amazon S3) bucket.

For complete source code and instructions on how to set up and run, see the full example on [GitHub](#).

Services used in this example

- Amazon Polly

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Create an application that analyzes customer feedback and synthesizes audio

The following code examples show how to create an application that analyzes customer comment cards, translates them from their original language, determines their sentiment, and generates an audio file from the translated text.

.NET

SDK for .NET

This example application analyzes and stores customer feedback cards. Specifically, it fulfills the need of a fictitious hotel in New York City. The hotel receives feedback from guests in various languages in the form of physical comment cards. That feedback is uploaded into the app through a web client. After an image of a comment card is uploaded, the following steps occur:

- Text is extracted from the image using Amazon Textract.
- Amazon Comprehend determines the sentiment of the extracted text and its language.
- The extracted text is translated to English using Amazon Translate.
- Amazon Polly synthesizes an audio file from the extracted text.

The full app can be deployed with the AWS CDK. For source code and deployment instructions, see the project in [GitHub](#).

Services used in this example

- Amazon Comprehend
- Lambda

- Amazon Polly
- Amazon Textract
- Amazon Translate

Java

SDK for Java 2.x

This example application analyzes and stores customer feedback cards. Specifically, it fulfills the need of a fictitious hotel in New York City. The hotel receives feedback from guests in various languages in the form of physical comment cards. That feedback is uploaded into the app through a web client. After an image of a comment card is uploaded, the following steps occur:

- Text is extracted from the image using Amazon Textract.
- Amazon Comprehend determines the sentiment of the extracted text and its language.
- The extracted text is translated to English using Amazon Translate.
- Amazon Polly synthesizes an audio file from the extracted text.

The full app can be deployed with the AWS CDK. For source code and deployment instructions, see the project in [GitHub](#).

Services used in this example

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

SDK for JavaScript (v3)

This example application analyzes and stores customer feedback cards. Specifically, it fulfills the need of a fictitious hotel in New York City. The hotel receives feedback from guests in various languages in the form of physical comment cards. That feedback is uploaded into

the app through a web client. After an image of a comment card is uploaded, the following steps occur:

- Text is extracted from the image using Amazon Textract.
- Amazon Comprehend determines the sentiment of the extracted text and its language.
- The extracted text is translated to English using Amazon Translate.
- Amazon Polly synthesizes an audio file from the extracted text.

The full app can be deployed with the AWS CDK. For source code and deployment instructions, see the project in [GitHub](#). The following excerpts show how the AWS SDK for JavaScript is used inside of Lambda functions.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });
```



```

const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

return {
  sentiment: Sentiment,
  language_code: languageCode,
};
};

```

```

import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
  eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/textract/latest/dg/API\_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );
};

```

```
    return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
 * sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
    const pollyClient = new PollyClient({});

    const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
        Engine: "neural",
        Text: sourceDestinationConfig.translated_text,
        VoiceId: "Ruth",
        OutputFormat: "mp3",
    });

    const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

    const audioKey = `${sourceDestinationConfig.object}.mp3`;

    // Store the audio file in S3.
    const s3Client = new S3Client();
    const upload = new Upload({
        client: s3Client,
        params: {
            Bucket: sourceDestinationConfig.bucket,
            Key: audioKey,
            Body: AudioStream,
            ContentType: "audio/mp3",
        },
    });

    await upload.done();
    return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Services used in this example

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

SDK for Ruby

This example application analyzes and stores customer feedback cards. Specifically, it fulfills the need of a fictitious hotel in New York City. The hotel receives feedback from guests in various languages in the form of physical comment cards. That feedback is uploaded into

the app through a web client. After an image of a comment card is uploaded, the following steps occur:

- Text is extracted from the image using Amazon Textract.
- Amazon Comprehend determines the sentiment of the extracted text and its language.
- The extracted text is translated to English using Amazon Translate.
- Amazon Polly synthesizes an audio file from the extracted text.

The full app can be deployed with the AWS CDK. For source code and deployment instructions, see the project in [GitHub](#).

Services used in this example

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Getting started with text-to-speech synthesis

The following code example shows how to:

- Clean up resources

Bash

AWS CLI with Bash script

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [Sample developer tutorials](#) repository.

```
#!/bin/bash

# Amazon Polly Getting Started Script
# This script demonstrates how to use Amazon Polly with the AWS CLI

set -euo pipefail

# Set up logging
LOG_FILE="polly-tutorial.log"
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
WORK_DIR=$(mktemp -d)
trap 'cleanup_temp' EXIT

cleanup_temp() {
    rm -rf "$WORK_DIR"
}

echo "Starting Amazon Polly tutorial at $(date)" > "$LOG_FILE"

# Function to log commands and their output
log_cmd() {
    echo "Running: $1" | tee -a "$LOG_FILE"
    # Use bash array to safely handle arguments
    bash -c "$1" 2>&1 | tee -a "$LOG_FILE" || return $?
}

# Function to check for errors
check_error() {
    if echo "$1" | grep -iq "error"; then
        echo "ERROR detected in output. Exiting script." | tee -a "$LOG_FILE"
        echo "$1" | tee -a "$LOG_FILE"
        return 1
    fi
    return 0
}

# Function to handle errors and cleanup
handle_error() {
    local line_number=$1
    echo "Error occurred at line $line_number. Attempting cleanup..." | tee -a
"$LOG_FILE"
    cleanup
    exit 1
}
```

```

}

# Function to clean up resources
cleanup() {
    echo "" | tee -a "$LOG_FILE"
    echo "===== " | tee -a
"$LOG_FILE"
    echo "CLEANUP PROCESS" | tee -a "$LOG_FILE"
    echo "===== " | tee -a
"$LOG_FILE"

    # Delete lexicon if it exists
    if [[ -n "${LEXICON_NAME:-}" ]]; then
        echo "Deleting lexicon: $LEXICON_NAME" | tee -a "$LOG_FILE"
        if aws polly delete-lexicon --name "$LEXICON_NAME" 2>&1 | tee -a
"$LOG_FILE"; then
            echo "Lexicon deleted successfully." | tee -a "$LOG_FILE"
        else
            echo "Warning: Failed to delete lexicon." | tee -a "$LOG_FILE"
        fi
    fi

    # Remove audio files
    for file in output.mp3 ssm1-output.mp3 lexicon-output.mp3 example.pls; do
        if [[ -f "$file" ]]; then
            rm -f "$file"
            echo "Removed $file" | tee -a "$LOG_FILE"
        fi
    done

    echo "Cleanup complete." | tee -a "$LOG_FILE"
}

# Trap errors with line number
trap 'handle_error ${LINENO}' ERR

# Verify AWS CLI is available
if ! command -v aws &> /dev/null; then
    echo "AWS CLI is not installed. Please install it first." | tee -a
"$LOG_FILE"
    exit 1
fi

# Verify AWS credentials are configured

```

```
if ! aws sts get-caller-identity &> /dev/null; then
    echo "AWS credentials are not configured. Please configure them first." | tee
    -a "$LOG_FILE"
    exit 1
fi

# Step 1: Verify Amazon Polly is available
echo "Step 1: Verifying Amazon Polly availability" | tee -a "$LOG_FILE"
if aws polly describe-voices --query 'Voices[0].Name' --output text &> /dev/null;
then
    echo "Amazon Polly is available. Proceeding with tutorial." | tee -a
    "$LOG_FILE"
else
    echo "Amazon Polly is not available in your AWS CLI installation or region."
    | tee -a "$LOG_FILE"
    echo "Please update your AWS CLI to the latest version or check your region."
    | tee -a "$LOG_FILE"
    exit 1
fi

# Step 2: List available voices
echo "" | tee -a "$LOG_FILE"
echo "Step 2: Listing available voices" | tee -a "$LOG_FILE"
log_cmd "aws polly describe-voices --language-code en-US --output text --query
'Voices[0:3].[Id, LanguageCode, Gender]'" || true

# Step 3: Basic text-to-speech conversion
echo "" | tee -a "$LOG_FILE"
echo "Step 3: Converting text to speech" | tee -a "$LOG_FILE"
OUTPUT_FILE="${WORK_DIR}/output.mp3"
POLLY_TEXT="Hello, welcome to Amazon Polly. This is a sample text to speech
conversion."
log_cmd "aws polly synthesize-speech --output-format mp3 --voice-id Joanna --text
'$POLLY_TEXT' '$OUTPUT_FILE'" || true

if [[ -f "$OUTPUT_FILE" ]]; then
    echo "Successfully created output.mp3 file." | tee -a "$LOG_FILE"
    echo "You can play this file with your preferred audio player." | tee -a
    "$LOG_FILE"
    cp "$OUTPUT_FILE" output.mp3
else
    echo "Failed to create output.mp3 file." | tee -a "$LOG_FILE"
    exit 1
fi
```

```

# Step 4: Using SSML for enhanced speech
echo "" | tee -a "$LOG_FILE"
echo "Step 4: Using SSML for enhanced speech" | tee -a "$LOG_FILE"
SSML_OUTPUT="${WORK_DIR}/ssml-output.mp3"
SSML_TEXT='<speack>Hello! <break time="1s"/> This is a sample of <emphasis>SSML
enhanced speech</emphasis>.</speack>'
log_cmd "aws polly synthesize-speech --output-format mp3 --voice-id Matthew --
text-type ssml --text '$SSML_TEXT' '$SSML_OUTPUT'" || true

if [[ -f "$SSML_OUTPUT" ]]; then
    echo "Successfully created ssml-output.mp3 file." | tee -a "$LOG_FILE"
    echo "You can play this file with your preferred audio player." | tee -a
"$LOG_FILE"
    cp "$SSML_OUTPUT" ssml-output.mp3
else
    echo "Failed to create ssml-output.mp3 file." | tee -a "$LOG_FILE"
    exit 1
fi

# Step 5: Working with lexicons
echo "" | tee -a "$LOG_FILE"
echo "Step 5: Working with lexicons" | tee -a "$LOG_FILE"

# Generate a random identifier for the lexicon (max 20 chars, alphanumeric only)
LEXICON_NAME="example$(openssl rand -hex 6 | cut -c 1-10)"
echo "Using lexicon name: $LEXICON_NAME" | tee -a "$LOG_FILE"

# Create a lexicon file
echo "Creating lexicon file..." | tee -a "$LOG_FILE"
LEXICON_FILE="${WORK_DIR}/example.pls"
cat > "$LEXICON_FILE" << 'EOF'
<?xml version="1.0" encoding="UTF-8"?>
<lexicon version="1.0"
    xmlns="http://www.w3.org/2005/01/pronunciation-lexicon"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2005/01/pronunciation-lexicon
    http://www.w3.org/TR/2007/CR-pronunciation-lexicon-20071212/pls.xsd"
    alphabet="ipa"
    xml:lang="en-US">
    <lexeme>
        <grapheme>AWS</grapheme>
        <alias>Amazon Web Services</alias>
    </lexeme>

```



```
</lexicon>
EOF

# Upload the lexicon
echo "Uploading lexicon..." | tee -a "$LOG_FILE"
LEXICON_ARN=$(aws polly put-lexicon --name "$LEXICON_NAME" --content
  file://"${LEXICON_FILE}" --query 'LexiconArn' --output text 2>&1) || true
if [[ -n "$LEXICON_ARN" && "$LEXICON_ARN" != "" && ! "$LEXICON_ARN" =~ error ]];
then
  echo "Lexicon uploaded with ARN: $LEXICON_ARN" | tee -a "$LOG_FILE"
  aws polly tag-resource --resource-arn "$LEXICON_ARN" --tags
    Key=project,Value=doc-smith Key=tutorial,Value=amazon-polly-gs 2>&1 | tee -a
    "$LOG_FILE" || true
else
  echo "Lexicon uploaded." | tee -a "$LOG_FILE"
fi

# List available lexicons
echo "Listing available lexicons..." | tee -a "$LOG_FILE"
log_cmd "aws polly list-lexicons --output text --query 'Lexicons[*].[Name]'" ||
  true

# Get details about the lexicon
echo "Getting details about the lexicon..." | tee -a "$LOG_FILE"
log_cmd "aws polly get-lexicon --name '$LEXICON_NAME' --output text --query
  'Lexicon.Name'" || true

# Use the lexicon when synthesizing speech
echo "Using the lexicon for speech synthesis..." | tee -a "$LOG_FILE"
LEXICON_OUTPUT="${WORK_DIR}/lexicon-output.mp3"
LEXICON_TEXT="I work with AWS every day."
log_cmd "aws polly synthesize-speech --output-format mp3 --voice-id Joanna --
  lexicon-names '$LEXICON_NAME' --text '$LEXICON_TEXT' '$LEXICON_OUTPUT'" || true

if [[ -f "$LEXICON_OUTPUT" ]]; then
  echo "Successfully created lexicon-output.mp3 file." | tee -a "$LOG_FILE"
  echo "You can play this file with your preferred audio player." | tee -a
    "$LOG_FILE"
  cp "$LEXICON_OUTPUT" lexicon-output.mp3
else
  echo "Failed to create lexicon-output.mp3 file." | tee -a "$LOG_FILE"
  exit 1
fi
```

```
# Summary of created resources
echo "" | tee -a "$LOG_FILE"
echo "===== " | tee -a
"$LOG_FILE"
echo "TUTORIAL SUMMARY" | tee -a "$LOG_FILE"
echo "===== " | tee -a
"$LOG_FILE"
echo "Created resources:" | tee -a "$LOG_FILE"
echo "1. Lexicon: $LEXICON_NAME" | tee -a "$LOG_FILE"
echo "2. Audio files:" | tee -a "$LOG_FILE"
echo "  - output.mp3" | tee -a "$LOG_FILE"
echo "  - ssm1-output.mp3" | tee -a "$LOG_FILE"
echo "  - lexicon-output.mp3" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"

# Cleanup with auto-confirmation
echo "" | tee -a "$LOG_FILE"
echo "===== " | tee -a
"$LOG_FILE"
echo "CLEANUP CONFIRMATION" | tee -a "$LOG_FILE"
echo "===== " | tee -a
"$LOG_FILE"
echo "Cleaning up all created resources..." | tee -a "$LOG_FILE"
cleanup

echo "" | tee -a "$LOG_FILE"
echo "Tutorial completed successfully!" | tee -a "$LOG_FILE"
echo "Log file: $LOG_FILE" | tee -a "$LOG_FILE"
```

- For API details, see the following topics in *AWS CLI Command Reference*.
 - [DeleteLexicon](#)
 - [DescribeVoices](#)
 - [GetLexicon](#)
 - [Help](#)
 - [ListLexicons](#)
 - [PutLexicon](#)
 - [SynthesizeSpeech](#)

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon Polly with an AWS SDK](#). This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon Polly

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

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

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the .
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You're also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

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

Topics

- [Data Protection in Amazon Polly](#)
- [Identity and Access Management in Amazon Polly](#)
- [Logging and Monitoring in Amazon Polly](#)
- [Compliance Validation for Amazon Polly](#)
- [Resilience in Amazon Polly](#)
- [Infrastructure Security in Amazon Polly](#)
- [Security Best Practices for Amazon Polly](#)
- [Using Amazon Polly with interface VPC endpoints](#)

Data Protection in Amazon Polly

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

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

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.

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

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

Encryption at Rest

Output of your Amazon Polly voice synthesis can be saved on your own system. You can also call Amazon Polly, and then encrypt the file with any encryption key of your choice and store it in Amazon Simple Storage Service (Amazon S3) or another secure storage. The Amazon Polly [SynthesizeSpeech](#) operation is stateless and is not associated with a customer identity. You can't retrieve it from Amazon Polly later.

Encryption in Transit

All text submissions are protected by TLS while in transit. Amazon Polly does not retain the content of text submissions.

Internetwork Traffic Privacy

Access to Amazon Polly is via the AWS console, CLI, or SDKs. Communications utilize Transport Layer Security (TLS) session encryption for confidentiality and [digital signatures](#) for authentication and integrity.

Identity and Access Management in Amazon Polly

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

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How Amazon Polly works with IAM](#)
- [Identity-based policy examples for Amazon Polly](#)
- [Amazon Polly API Permissions: Actions, Permissions, and Resources Reference](#)
- [Troubleshooting Amazon Polly identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see [Troubleshooting Amazon Polly identity and access](#))
- **Service administrator** - determine user access and submit permission requests (see [How Amazon Polly works with IAM](#))

- **IAM administrator** - write policies to manage access (see [Identity-based policy examples for Amazon Polly](#))

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more

information, see [Require human users to use federation with an identity provider to access AWS using temporary credentials](#) in the *IAM User Guide*.

An [IAM group](#) specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity with specific permissions that provides temporary credentials. You can assume a role by [switching from a user to an IAM role \(console\)](#) or by calling an AWS CLI or AWS API operation. For more information, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between

managed and inline policies, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must [specify a principal](#) in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see [Service control policies](#) in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see [Resource control policies \(RCPs\)](#) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How Amazon Polly works with IAM

Before you use IAM to manage access to Amazon Polly, learn what IAM features are available to use with Amazon Polly.

IAM features you can use with Amazon Polly

IAM feature	Amazon Polly support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	No
ACLs	No
ABAC (tags in policies)	No
Temporary credentials	Yes
Forward access sessions (FAS) for Amazon Polly	Yes
Service roles	No
Service-linked roles	No

To get a high-level view of how Amazon Polly and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for Amazon Polly

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can

perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for Amazon Polly

To view examples of Amazon Polly identity-based policies, see [Identity-based policy examples for Amazon Polly](#).

Resource-based policies within Amazon Polly

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for Amazon Polly

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Polly actions, see [Actions defined by Amazon Polly](#) in the *Service Authorization Reference*.

Policy actions in Amazon Polly use the following prefix before the action:

```
polly
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
    "polly:action1",  
    "polly:action2"  
]
```

To view examples of Amazon Polly identity-based policies, see [Identity-based policy examples for Amazon Polly](#).

Policy resources for Amazon Polly

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon Polly resource types and their ARNs, see [Resources defined by Amazon Polly](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon Polly](#).

To view examples of Amazon Polly identity-based policies, see [Identity-based policy examples for Amazon Polly](#).

Policy condition keys for Amazon Polly

Supports service-specific policy condition keys: No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of Amazon Polly condition keys, see [Condition keys for Amazon Polly](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon Polly](#).

To view examples of Amazon Polly identity-based policies, see [Identity-based policy examples for Amazon Polly](#).

ACLs in Amazon Polly

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon Polly

Supports ABAC (tags in policies): No

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [Define permissions with ABAC authorization](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

Using temporary credentials with Amazon Polly

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#) and [AWS services that work with IAM](#) in the *IAM User Guide*.

Cross-service forward access sessions (FAS) for Amazon Polly

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for Amazon Polly

Supports service roles: No

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break Amazon Polly functionality. Edit service roles only when Amazon Polly provides guidance to do so.

Service-linked roles for Amazon Polly

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Amazon Polly IAM roles

You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. 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. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see [Access Management](#) in the *IAM User Guide*.

The following is an example policy that grants permissions to put and get lexicons as well as to list those lexicons currently available.

Amazon Polly supports Identity-based policies for actions at the resource-level. In some cases, the resource can be limited by an ARN. This is true for the SynthesizeSpeech, StartSpeechSynthesisTask, PutLexicon, GetLexicon, and DeleteLexicon operations. In these cases, the Resource value is indicated by the ARN. For example: `arn:aws:polly:us-east-2:account-id:lexicon/*` as the Resource value specifies permissions on all owned lexicons within the us-east-2 Region.

However, not all operations use ARNs. This is the case with the DescribeVoices, ListLexicons, GetSpeechSynthesisTasks, and ListSpeechSynthesisTasks operations.

For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

Identity-based policy examples for Amazon Polly

By default, users and roles don't have permission to create or modify Amazon Polly resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Create IAM policies \(console\)](#) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon Polly, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon Polly](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the Amazon Polly console](#)
- [Allow users to view their own permissions](#)
- [AWS managed \(predefined\) policies for Amazon Polly](#)
- [Amazon Polly updates to AWS managed policies](#)
- [Customer-managed policy examples](#)

Policy best practices

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

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more

information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the Amazon Polly console

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

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Amazon Polly console, also attach the Amazon Polly *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

To use the Amazon Polly console, grant permissions to all the Amazon Polly APIs. There are no additional permissions needed. To get full console functionality you can use following policy:

Allow users to view their own permissions

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

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

AWS managed (predefined) policies for Amazon Polly

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 Polly:

- **AmazonPollyReadOnlyAccess** – Grants read-only access to resources, allows listing lexicons, fetching lexicons, listing available voices and synthesizing speech (including, applying lexicons to the synthesized speech).

To view more details about the policy, including the latest version of the JSON policy document, see [AmazonPollyReadOnlyAccess](#) in the *AWS Managed Policy Reference Guide*.

- **AmazonPollyFullAccess** – Grants full access to resources and all the supported operations.

To view more details about the policy, including the latest version of the JSON policy document, see [AmazonPollyFullAccess](#) in the *AWS Managed Policy Reference Guide*.

Note

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

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

Amazon Polly updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Polly since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Polly Document History page.

Amazon Polly updates to AWS managed policies

Policy	Change	Date
AmazonPollyReadOnlyAccess	Updated managed policy - Added permission for <code>polly:StartSpeechSynthesisS</code>	March 19, 2026

Policy	Change	Date
	stream to support bidirectional streaming speech synthesis capabilities.	

Customer-managed policy examples

In this section, you can find example user policies that grant permissions for various Amazon Polly actions. These policies work when you're using AWS SDKs or the AWS CLI. When you're using the console, grant permissions to all the Amazon Polly APIs.

Note

All examples use the us-east-2 Region and contain fictitious account IDs.

Examples

- [Example 1: Allow All Amazon Polly Actions](#)
- [Example 2: Allow all Amazon Polly actions except DeleteLexicon](#)
- [Example 3: Allow DeleteLexicon](#)
- [Example 4: Allow Delete Lexicon in a specified Region](#)
- [Example 5: Allow DeleteLexicon for specified Lexicon](#)

Example 1: Allow All Amazon Polly Actions

After you sign up (see [Getting started with Amazon Polly](#)) create an administrator user to manage your account, including creating users and managing their permissions.

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

JSON

```
{
```

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

Example 2: Allow all Amazon Polly actions except DeleteLexicon

The following permissions policy grants the user permissions to perform all actions except DeleteLexicon, with the permissions for delete explicitly denied in all Regions.

Example 3: Allow DeleteLexicon

The following permissions policy grants the user permissions to delete any lexicon that you own regardless of the project or Region in which it is located.

JSON

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

Example 4: Allow Delete Lexicon in a specified Region

The following permissions policy grants the user permissions to delete any lexicon in any project that you own that is located in a single Region (in this case, us-east-2).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowDeleteSpecifiedRegion",
    "Effect": "Allow",
    "Action": [
      "polly:DeleteLexicon"],
    "Resource": "arn:aws-eusc:polly:us-east-2:111122223333:lexicon/*"
  ]
}
```

Example 5: Allow DeleteLexicon for specified Lexicon

The following permissions policy grants the user permissions to delete a specific lexicon that you own (in this case, myLexicon) in a specific Region (in this case, us-east-2).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowDeleteForSpecifiedLexicon",
    "Effect": "Allow",
    "Action": [
      "polly:DeleteLexicon"],
    "Resource": "arn:aws-eusc:polly:us-east-2:111122223333:lexicon/myLexicon"
  ]
}
```

Amazon Polly API Permissions: Actions, Permissions, and Resources Reference

When you're setting up a permissions policy that you can attach to an IAM identity (identity-based policies), you can use the following list as a reference. The list includes each Amazon Polly API

operation, the corresponding actions 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.

You can use AWS-wide condition keys in your Amazon Polly policies to express conditions. For a complete list of AWS-wide keys, see [available keys](#) in the *IAM User Guide*.

Note

To specify an action, use the polly prefix followed by the API operation name (for example, polly:GetLexicon).

Amazon Polly supports Identity-based policies for actions at the resource-level. Therefore, the Resource value is indicated by the ARN. For example: `arn:aws:polly:us-east-2:account-id:lexicon/*` as the Resource value specifies permissions on all owned lexicons within the us-east-2 Region.

Because Amazon Polly doesn't support permissions for actions at the resource-level, most policies specify a wildcard character (*) as the Resource value. However, if it is necessary to limit permissions to a specific Region this wildcard character is replaced with the appropriate ARN: `arn:aws:polly:region:account-id:lexicon/*`.

Amazon Polly API and Required Permissions for Actions

API Operation: [DeleteLexicon](#)

Required Permissions (API Action): polly:DeleteLexicon

Resources: `arn:aws:polly:region:account-id:lexicon/LexiconName`

API Operation: [DescribeVoices](#)

Required Permissions (API Action): polly:DescribeVoices

Resources: `arn:aws:polly:region:account-id:lexicon/voice-name`

API Operation: [GetLexicon](#)

Required Permissions (API Action): polly:GetLexicon

Resources: arn:aws:polly:*region*:*account-id*:lexicon/*voice-name*

API Operation: [ListLexicons](#)

Required Permissions (API Action): polly:ListLexicons

Resources: arn:aws:polly:*region*:*account-id*:lexicon/*

API Operation: [PutLexicon](#)

Required Permissions (API Action): polly:ListLexicons

Resources: *

API Operation: [SynthesizeSpeech](#)

Required Permissions (API Action): polly:SynthesizeSpeech

Resources: *

Troubleshooting Amazon Polly identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Polly and IAM.

Topics

- [I am not authorized to perform an action in Amazon Polly](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my Amazon Polly resources](#)

I am not authorized to perform an action in Amazon Polly

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a fictional *my-example-widget* resource but doesn't have the fictional polly:*GetWidget* permissions.

```
User: arn:aws-eusc:iam::123456789012:user/mateojackson is not authorized to perform:
polly:GetWidget on resource: my-example-widget
```


In this case, the policy for the `mateojackson` user must be updated to allow access to the *my-example-widget* resource by using the `polly:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon Polly.

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

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

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

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon Polly resources

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

To learn more, consult the following:

- To learn whether Amazon Polly supports these features, see [How Amazon Polly works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Logging and Monitoring in Amazon Polly

Monitoring is an important part of maintaining the reliability, availability, and performance of your Amazon Polly applications. To monitor Amazon Polly API calls, you can use AWS CloudTrail. To monitor the status of your jobs, use Amazon CloudWatch Logs.

- **Amazon CloudWatch Alarms** – Using CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon Simple Notification Service topic or AWS Auto Scaling policy. CloudWatch alarms don't invoke actions when a metric is in a particular state. Rather the state must have changed and been maintained for a specified number of periods. For more information, see [Integrating CloudWatch with Amazon Polly](#).
- **CloudTrail logs** – CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon Polly. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Polly. You can also determine the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see [Logging Amazon Polly API calls with AWS CloudTrail](#).

Compliance Validation for Amazon Polly

Third-party auditors assess the security and compliance of Amazon Polly as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see . For general information, see .

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

- – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

Resilience in Amazon Polly

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

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Infrastructure Security in Amazon Polly

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

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

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

Security Best Practices for Amazon Polly

Your trust, privacy, and the security of your content are our highest priorities. We implement responsible and sophisticated technical and physical controls designed to prevent unauthorized access to, or disclosure of, your content and ensure that our use complies with our commitments to you. For more information, see [AWS Data Privacy FAQ](#).

Amazon Polly does not retain the content of text submissions.

For a broad view of AWS security, including compliance, penetration testing, bulletins, and resources, visit the [AWS Cloud Security](#) website.

Using Amazon Polly with interface VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and Amazon Polly. You can use this connection to synthesize speech with Amazon Polly without traversing the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. To connect your VPC to Amazon Polly, you define an *interface VPC endpoint* for Amazon Polly. This type of endpoint enables you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to Amazon Polly without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see the [What is Amazon VPC](#) in the *Amazon VPC User Guide*.

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IP addresses. For more information, see [New - AWS PrivateLink for AWS services](#).

The following steps are for users of Amazon VPC. For more information, see [Getting Started](#) in the *Amazon VPC User Guide*.

Availability

VPC endpoints are supported in all the [Regions where Amazon Polly is supported](#). For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

FIPS endpoints are available in the following regions:

- US East (N. Virginia) (us-east-1)
- US East (Ohio) (us-east-2)
- US West (N. California) (us-west-1)
- US West (Oregon) (us-west-2)
- AWS GovCloud (US-West) (us-gov-west-1)

- Canada (Central) (ca-central-1)

The FIPS endpoints are of the form `com.amazonaws.REGION.polly-fips`.

Creating a VPC endpoint for Amazon Polly

To start using Amazon Polly with your VPC, create an interface VPC endpoint for Amazon Polly. The service to choose is **com.amazonaws.*Region*.polly**. You don't need to change any settings for Amazon Polly. For more information, see [Creating an Interface Endpoint](#) in the *Amazon VPC User Guide*.

Testing the connection between your VPC and Amazon Polly

After you create the endpoint, you can test the connection.

To test the connection between your VPC and your Amazon Polly endpoint

1. Connect to an Amazon EC2 instance that resides in your VPC. For information about connecting, see [Connect to your Linux instance](#) or [Connecting to your Windows instance](#) in the Amazon EC2 documentation.
2. From the instance, use `aws polly describe-voices` from the AWS CLI to list available Amazon Polly voices.

If the response to the command includes the list of available Amazon Polly voices, the command has succeeded, and your VPC endpoint is working.

Controlling access to your Amazon Polly endpoint

A VPC endpoint policy is an IAM resource policy that you attach to an endpoint when you create or modify the endpoint. If you don't attach a policy when you create an endpoint, we attach a default policy for you that allows full access to the service. An endpoint policy doesn't override or replace IAM user policies or service-specific policies. It's a separate policy for controlling access from the endpoint to the specified service.

Endpoint policies must be written in JSON format.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following is an example of an endpoint policy for Amazon Polly. This policy enables users connecting to Amazon Polly through the VPC to describe voices and synthesize speech with Amazon Polly, and prevents them from performing other Amazon Polly actions.

```
{
  "Statement": [
    {
      "Sid": "SynthesisAndDescribeVoicesOnly",
      "Principal": "*",
      "Action": [
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

To modify the VPC endpoint policy for Amazon Polly

1. Open the Amazon VPC console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/vpc>.
2. In the navigation pane, choose **Endpoints**.
3. If you have not already created the endpoint for Amazon Polly, choose **Create endpoint**. Then select **com.amazonaws.*Region*.polly** and choose **Create endpoint**.
4. Select the **com.amazonaws.*Region*.polly** endpoint, and choose the **Policy** tab in the lower half of the screen.
5. Choose **Edit Policy** and make the changes to the policy.

Support for VPC context keys

Amazon Polly supports the `aws:SourceVpc` and `aws:SourceVpce` context keys that can limit access to specific VPCs or specific VPC endpoints. These keys work only when the user is using VPC endpoints. For more information, see [Keys Available for Some Services](#) in the *IAM user Guide*.

Logging Amazon Polly API calls with AWS CloudTrail

Amazon Polly is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Polly. CloudTrail captures all API calls for Amazon Polly as events. The calls captured include calls from the Amazon Polly console and code calls to the Amazon Polly API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Polly. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Polly, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](#).

Amazon Polly information in CloudTrail

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

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

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

Amazon Polly supports logging the following actions as events in CloudTrail log files:

- [DeleteLexicon](#)
- [DescribeVoices](#)
- [GetLexicon](#)
- [GetSpeechSynthesisTask](#)
- [ListLexicons](#)
- [ListSpeechSynthesisTasks](#)
- [PutLexicon](#)
- [StartSpeechSynthesisTask](#)
- [SynthesizeSpeech](#)

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

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

For more information, see the [CloudTrail userIdentity Element](#).

Example: Amazon Polly Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

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

```
{
  "Records": [
    {
      "awsRegion": "us-east-2",
      "eventID": "19bd70f7-5e60-4cdc-9825-936c552278ae",
```



```

    "eventName": "SynthesizeSpeech",
    "eventSource": "polly.amazonaws.com",
    "eventTime": "2016-11-02T03:49:39Z",
    "eventType": "AwsApiCall",
    "eventVersion": "1.05",
    "recipientAccountId": "123456789012",
    "requestID": "414288c2-a1af-11e6-b17f-d7cfc06cb461",
    "requestParameters": {
"lexiconNames": [
    "SampleLexicon"
],
    "engine": "neural",
    "outputFormat": "mp3",
    "sampleRate": "22050",
    "text": "*****",
    "textType": "text",
    "voiceId": "Kendra"
},
    "responseElements": null,
    "sourceIPAddress": "1.2.3.4",
    "userAgent": "Amazon CLI/Polly 1.10 API 2016-06-10",
    "userIdentity": {
"accessKeyId": "EXAMPLE_KEY_ID",
    "accountId": "123456789012",
    "arn": "arn:aws:iam::123456789012:user/Alice",
    "principalId": "EX_PRINCIPAL_ID",
    "type": "IAMUser",
    "userName": "Alice"
    }
    }
}

```

Integrating CloudWatch with Amazon Polly

When you interact with Amazon Polly, it sends the following metrics and dimensions to CloudWatch every minute. You can use the following procedures to view the metrics for Amazon Polly.

You can monitor Amazon Polly using CloudWatch, which collects and processes raw data from Amazon Polly into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, Amazon Polly metric data is sent to CloudWatch in 1 minute intervals. For more information, see [What Is Amazon CloudWatch](#) in the *Amazon CloudWatch User Guide*.

Getting CloudWatch Metrics (Console)

1. Open the CloudWatch console at <https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the **CloudWatch Metrics by Category** pane, under the metrics category for Amazon Polly, select a metrics category, and then in the upper pane, scroll down to view the full list of metrics.

Getting CloudWatch metrics on the AWS CLI

The following code display available metrics for Amazon Polly.

```
aws cloudwatch list-metrics --namespace "AWS/Polly"
```

The preceding command returns a list of Amazon Polly metrics similar to the following. The `MetricName` element identifies what the metric is.

```
{
  "Metrics": [
    {
      "Namespace": "AWS/Polly",
      "Dimensions": [
        {
```

```
        "Name": "Operation",
        "Value": "SynthesizeSpeech"
    }
],
"MetricName": "ResponseLatency"
},
{
    "Namespace": "AWS/Polly",
    "Dimensions": [
        {
            "Name": "Operation",
            "Value": "SynthesizeSpeech"
        }
    ],
    "MetricName": "RequestCharacters"
}
```

For more information, see [GetMetricStatistics](#) in the *Amazon CloudWatch API Reference*.

Amazon Polly Metrics

Amazon Polly produces the following metrics for each request. These metrics are aggregated and in one minute intervals sent to CloudWatch where they are available.

Metric	Description
RequestCharacters	The number of characters in the request. This is billable characters only and does not include SSML tags. Valid Dimension: Operation Valid Statistics: Minimum, Maximum, Average, SampleCount, Sum Unit: Count
ResponseLatency	The latency between when the request was made and the start of the streaming response. Valid Dimensions: Operation

Metric	Description
	Valid Statistics: Minimum, Maximum, Average, SampleCount Unit: microseconds
2XXCount	HTTP 200 level code returned upon a successful response. Valid Dimensions: Operation Valid Statistics: Average, SampleCount, Sum Unit: Count
4XXCount	HTTP 400 level error code returned when Amazon Polly rejects a request because of a client-side problem rather than a server-side error. Client-side problems include malformed or invalid input in the request. For example, a lexicon name in a PutLexicon request exceeds the maximum allowed length. Other causes include an invalid parameter value or an otherwise malformed request. For each successful response, a zero (0) is emitted. Valid Dimensions: Operation Valid Statistics: Average, SampleCount, Sum Unit: Count

Metric	Description
5XXCount	HTTP 500 level error code returned when a request fails because of a server-side problem on the Amazon Polly service. For each successful response, a zero (0) is emitted. Valid Dimensions: Operation Valid Statistics: Average, SampleCount, Sum Unit: Count

Dimensions for Amazon Polly Metrics

Amazon Polly metrics use the *AWS/Polly* namespace and provide metrics for the following dimension:

Dimension	Description
Operation	Metrics are grouped by the API method they refer to. Possible values are <code>SynthesizeSpeech</code> , <code>PutLexicon</code> , <code>DescribeVoices</code> , etc.

Amazon Polly API Reference

The API reference is now a separate document. For more information, see [Amazon Polly API Reference](#).

Document History for Amazon Polly

The following table describes important changes in each release of the *Amazon Polly Developer Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** August 12 2026

Change	Description	Date
Clarified CloudWatch metrics descriptions	The Amazon Polly CloudWatch metric descriptions now clarify that 4XXCount reflects client-side errors and 5XXCount reflects server-side errors. For more information, see Integrating CloudWatch with Amazon Polly .	September 15, 2026
Amazon Polly generative voices now available in Asia Pacific (Sydney)	Amazon Polly generative voices and bidirectional streaming are now available in an additional AWS Region: Asia Pacific (Sydney). For more information, see Generative voices .	August 12, 2026
New region added for neural voices	Amazon Polly is now available in the Asia Pacific (Thailand) AWS Region. This Region supports neural TTS (NTTS) voices. For more information, see Neural voices .	May 28, 2026
New region added for generative voices	Amazon Polly generative voices and bidirectional streaming are now available in an additional AWS Region:	May 28, 2026

Europe (Zurich). For more information, see [Generative voices](#).

[Bidirectional streaming available in new regions](#)

Amazon Polly bidirectional streaming is now available in two additional AWS Regions: Europe (London) and Canada (Central). For more information, see [Generative voices](#).

April 20, 2026

[Updated managed policy](#)

Amazon Polly has updated the managed policy `AmazonPollyReadOnlyAccess` to include permission for bidirectional streaming speech synthesis. For information, see [Amazon Polly updates to AWS managed policies](#).

March 19, 2026

[New voices added for generative text-to-speech](#)

Amazon Polly now provides ten additional generative voices: Brian, Aria, Jasmine, Tiffany, Ambre, Florian, Sabrina, Lennart, Beatrice, Lorenzo. See [Generative voices](#) for a list of generative TTS voices.

March 19, 2026

[New regions added for generative voices](#)

Amazon Polly generative voices are now available in two additional AWS Regions: Europe (London), and Canada (Central). For more information, see [Generative voices](#).

March 19, 2026

New regions added for generative voices	Amazon Polly generative voices are now available in three additional AWS Regions: Asia Pacific (Tokyo), Asia Pacific (Seoul), and Asia Pacific (Singapore). For more information, see Generative voices .	November 18, 2025
New voices added for generative text-to-speech	Amazon Polly now provides six additional generative voices: Seoyeon, Camila, Hannah, Niamh, Laura, and Lisa. See Generative voices for a list of generative TTS voices.	November 14, 2025
New region added for neural voices	Amazon Polly is now available in the Europe (Zurich) AWS Region. This Region supports neural TTS (NTTS) voices. For more information, see Neural voices .	October 20, 2025
New voices added for generative text-to-speech	Amazon Polly now provides seven additional generative voices: Salli, Isabelle, Céline, Liam, Gabrielle, Ola, and Ewa. See Generative voices for a list of generative TTS voices.	August 26, 2025
New region added for neural and standard voices	Amazon Polly is now available in the Asia Pacific (Malaysia) AWS Region. This Region supports neural TTS (NTTS) and standard voices. For more information, see Neural voices and Standard voices .	March 27, 2025

New voice added for neural text-to-speech	Amazon Polly now provides one additional Korean voice: Jihye. See Neural voices for a list of NTTS voices.	March 26, 2025
New region added for neural and standard voices	Amazon Polly is now available in the Europe (Spain) AWS Region. This Region supports neural TTS (NTTS) and standard voices. For more information, see Neural voices and Standard voices .	February 18, 2025
New voice added for neural text-to-speech	Amazon Polly now provides one additional English (Singaporean) voice: Jasmine. See Neural voices for a list of NTTS voices.	February 11, 2025
New voices added for generative text-to-speech	Amazon Polly now provides seven new generative voices: Pedro, Andrés, Sergio, Daniel, Kajal, Rémi, Bianca. See Generative voices for a list of generative TTS voices.	November 21, 2024
New long-form voices added	Amazon Polly now provides more long-form voices. One additional English (US) and two new Spanish (Spain) long-form voices were added: Patrick, Alba, and Raúl. See Long-form voices for a list of all long-form voices.	November 14, 2024

New voices added for generative text-to-speech	Amazon Polly now provides six new generative voices: Ayanda, Léa, Lucia, Mía, Lupe, and Vicki. See Generative voices for a list of generative TTS voices.	November 6, 2024
New voices added for generative text-to-speech	Amazon Polly now provides four new generative voices: Olivia, Danielle, Joanna, and Stephen. See Generative voices for a list of generative TTS voices.	October 10, 2024
New languages added for text-to-speech	Amazon Polly now provides two new TTS languages: Czech (cs-CZ) and German (Swiss) (de-CH). See Supported languages for a list of languages.	September 26, 2024
New voices added for NTTTS	Amazon Polly now provides two new NTTTS voices: Jitka and Sabrina. See Neural voices for a list of NTTTS voices.	August 27, 2024
New generative voice engine added	Amazon Polly now offers a generative voice engine designed for longer content, with three English voices in a generative variant: Amy, Matthew, and Ruth. See Generative voices for more information.	March 28, 2024

New voice added for NTTS	Amazon Polly now provides the NTTS Turkish voice Burcu. See Neural voices for a list of NTTS voices.	February 14, 2024
New long-form voice engine added	Amazon Polly now offers a long-form voice engine designed for longer content, with three en-US voices: Danielle, Gregory, and Ruth. See Long-form voices for more information.	November 16, 2023
New voices added for NTTS	Amazon Polly now provides two new NTTS US English voices: Danielle and Gregory. See Neural voices for a list of NTTS voices.	October 5, 2023
Amazon Polly for Windows	The Amazon Polly Windows Speech Application Programming Interface (SAPI) plugin will no longer be supported.	September 26, 2023
Updated quota guidance for Amazon Polly	Updated Amazon Polly quotas guide. Added examples and clarification of terms. Refer to Quotas in Amazon Polly for the updates.	August 17, 2023
New voice added for NTTS	Amazon Polly now provides the Gulf Arabic NTTS voice Zayd. See Neural voices for a list of NTTS voices.	August 16, 2023

New voice added for NTTS	Amazon Polly now provides the Belgian French NTTS voice Isabelle. See Neural voices for a list of NTTS voices.	August 1, 2023
New voice added for NTTS	Amazon Polly now provides the Belgian Dutch (Flemish) NTTS voice Lisa. See Neural voices for a list of NTTS voices.	June 7, 2023
New voices added for NTTS	Amazon Polly now provides two new NTTS voices: Irish English (Niamh), and Danish (Sofie). See Neural voices for a list of NTTS voices.	May 30, 2023
Updated the IAM guidance for Amazon Polly	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM .	April 19, 2023
WordPress update	The Amazon Polly WordPress plugin will no longer be supported.	April 6, 2023
New Region added	Amazon Polly is now available in the Asia Pacific (Osaka) AWS Region. This Region supports neural TTS (NTTS). For more information, see Feature and Region Compatibility for a list of regions that support NTTS.	April 5, 2023

New voices added for NTTS	Amazon Polly now provides two new Japanese NTTS voices: Kazuha and Tomoko. See Neural voices for a list of NTTS voices.	February 7, 2023
New voices added for NTTS	Amazon Polly now provides two new US English NTTS voices: Stephen and Ruth. See Neural voices for a list of NTTS voices.	January 31, 2023
New voices added for NTTS	Amazon Polly now provides new NTTS voices for: Brazilian Portuguese (Thiago), Castilian Spanish (Sergio), French (Rémi), Italian (Adriano), and Mexican Spanish (Andrés). See Neural voices for a list of NTTS voices.	January 24, 2023
New voices added for NTTS	Amazon Polly now provides NTTS voices for Arabic (Hala) and Polish (Ola). See Neural voices for a list of NTTS voices.	November 17, 2022
Release AWS PrivateLink support	Amazon Polly now provides AWS PrivateLink support. See Using Amazon Polly with VPC endpoints to learn more.	November 9, 2022
New voices and languages added for NTTS	Amazon Polly now provides NTTS voices for Finnish (Suvi), Norwegian (Ida), and Swedish (Elin). See Neural voices for a list of NTTS voices.	November 8, 2022

New voice added for NTTS	Amazon Polly now provides the Dutch NTTS voice Laura. See Neural voices for a list of NTTS voices.	November 2, 2022
New Region added	Amazon Polly is now available in the Europe (Paris) AWS Region. This Region supports neural TTS (NTTS). For more information, see Feature and Region Compatibility for a list of regions that support NTTS.	September 22, 2022
New voice and language added for NTTS	Amazon Polly now provides the Cantonese NTTS voice Hiujin. See Neural voices for a list of NTTS voices.	September 20, 2022
New Region added	Amazon Polly is now available in the Asia Pacific (Mumbai) AWS Region. This Region supports neural TTS (NTTS). For more information, see Feature and Region Compatibility for a list of regions that support NTTS.	September 1, 2022
New voice added for NTTS	Amazon Polly now provides the Mandarin voice Zhiyu as an NTTS voice. See Neural voices for a list of NTTS voices.	August 23, 2022
New voice added for NTTS	Amazon Polly now provides the Hindi NTTS voice Kajal. See Neural voices for a list of NTTS voices.	July 27, 2022

[New voices added for NTTS](#)

Amazon Polly now provides NTTS voices for US Spanish (Pedro), German (Daniel), Canadian French (Liam), and UK English (Arthur). See [Neural voices](#) for a list of NTTS voices.

June 28, 2022

[New voice added for NTTS](#)

Amazon Polly now provides the Portuguese (Brazilian) voice Vitória as an NTTS voice. See [Neural voices](#) for a list of NTTS voices.

April 27, 2022

[New voice added for NTTS](#)

Amazon Polly now provides the Portuguese (European) voice Inês as an NTTS voice. See [Neural voices](#) for a list of NTTS voices.

April 26, 2022

[New voice and language added for NTTS](#)

Amazon Polly now provides the German (Austrian) language and the NTTS voice Hannah. See [Neural voices](#) for a list of NTTS voices.

April 19, 2022

[New voices and language added for NTTS](#)

Amazon Polly now provides the Spanish (Mexican) voice Mia as an NTTS voice. A new language, Catalan, was added along with the NTTS voice Arlet. See [Neural voices](#) for a list of NTTS voices.

March 22, 2022

New voice added for NTTS	Amazon Polly now provides the Japanese voice Takumi as an NTTS voice. See Neural voices for a list of NTTS voices.	December 6, 2021
New voice added for NTTS	Amazon Polly now provides the French voice Léa as an NTTS voice. See Neural voices for a list of NTTS voices.	November 18, 2021
New voices added for NTTS	Amazon Polly now provides the Italian voice Bianca and the European Spanish voice Lucia as NTTS voices. See Neural voices for a list of NTTS voices.	November 8, 2021
New voice added for NTTS	Amazon Polly now provides a new South African English voice, Ayanda. The voice is available as an NTTS voice only. See Neural voices for a list of NTTS voices.	September 1, 2021
New Region added	Amazon Polly is now available in the Africa (Cape Town) AWS Region. This Region supports neural TTS (NTTS). For more information, see Feature and Region Compatibility for a list of regions that support NTTS.	September 1, 2021

[New language and voice added](#)

Amazon Polly now supports New Zealand English (en-NZ). A new NTTS voice, Aria, speaks New Zealand English and a selection of Maori words.

August 24, 2021

[New feature](#)

Amazon Polly makes the conversational speaking style the default version for the neural Matthew and Joanna voices. We removed references to the conversational speaking style.

June 28, 2021

[New voice added for NTTS](#)

Amazon Polly now provides the German voice Vicki as an NTTS voice.

June 15, 2021

[New voice added](#)

A new female voice, Gabrielle, has been added to the French (Canadian) (fr-CA) locale. The voice is high quality and only available as an NTTS voice. Like all neural voices, it is only available in certain regions. For a list of regions, see [Feature and region compatibility](#).

June 1, 2021

[New voice added for NTTS](#)

Amazon Polly now provides the Korean voice Seoyeon as an NTTS voice.

May 11, 2021

[New Region added for NTTS](#)

Amazon Polly now supports neural TTS (NTTS) in the Canada (Central) AWS Region. For more information, see [Feature and Region Compatibility](#) for NTTS.

March 17, 2021

[New voice available for newscaster style](#)

In addition to the Matthew, Joanna, and Lupe voices for the Newscaster speaking style, Amazon Polly now provides an additional option for this speaking style. Using the neural engine, you can use the Amy voice in British English for the Newscaster style. For more information, see [NTTS Speaking Styles](#).

November 10, 2020

[New Regions added for NTTS](#)

In addition to the existing Regions for NTTS (us-east-1, us-west-2, eu-west-1, and ap-southeast-2), neural voices are now supported in four additional Regions: (ap-north-east-1 (Tokyo), ap-southeast-1 (Singapore), eu-central-1 (Frankfurt), and eu-west-2 (London). For more information, see [Feature and Region Compatibility](#) for NTTS.

September 3, 2020

[New voice added](#)

In addition to child voices Ivy and Justin, a new male child voice, Kevin, has been added to American English (en-US). This new voice is very high quality and is only available as an NTTS voice. Like all neural voices, it is only supported in four Regions: us-east-1 (N. Virginia), us-west-2 (Oregon), eu-west-1 (Ireland), and ap-southeast-2 (Sydney). For more information, see [NTTS Voices](#).

June 16, 2020

[New voice available for newscaster style](#)

In addition to the Matthew and Joanna voices for the Newscaster speaking style, Amazon Polly now provides an additional option for this speaking style. Using the neural engine, you can use the Lupe voice in Spanish (American) for the Newscaster style. For more information, see [NTTS Speaking Styles](#).

April 16, 2020

New feature

In addition to the Newscaster speaking style, Amazon Polly now provides a second NTTS speaking style to help you synthesize even better text to speech passages. The Conversational style uses the neural system to generate speech in a more friendly and expressive conversational style that can be used in many use cases. For more information, see [NTTS Speaking Styles](#).

November 25, 2019

New voices added

Two new voices added: Camila (female, Portuguese-Brazil) and Lupe (female, Spanish-US).

October 23, 2019

New feature added

Addition of [Amazon Polly for Windows plugin](#) to incorporate the full range of Amazon Polly voices into Windows SAPI-compliant applications.

September 26, 2019

Major new feature	In addition to the standard text-to-speech (TTS) voices supported by Amazon Polly since its launch, Amazon Polly now provides an improved Neural TTS (NTTS) system that can provide even higher quality voices, thereby providing you with the most natural and human-like text-to-speech voices possible. For more information, see Neural Text-to-Speech .	July 30, 2019
New voices added	New voices added: Lucia (female, Spanish), and Bianca (female, Italian).	August 2, 2018
New language added	New language added: Mexican Spanish (es-MX). This language uses the female voice of Mia.	August 2, 2018
New language added	New language added: Hindi (hi-IN). This voice uses the female voice of Aditi, which is also used for Indian English, making Aditi Amazon Polly's first bilingual voice.	August 2, 2018
New feature added	Addition of Speech synthesis of long text passages (up to 100,000 billed characters).	July 17, 2018
New SSML feature added	Addition of Maximum Duration for Synthesized Speech .	July 17, 2018

New voice added	New voice added: Léa (female, French).	June 5, 2018
Region expansion	Expansion of Amazon Polly service to all commercial regions.	June 4, 2018
New language added	New language added: Korean (ko-KR).	June 4, 2018
Expanded feature	The Amazon Polly WordPress Plugin feature, including addition of Amazon Translate capabilities.	June 4, 2018
New voices added	Two new voices added: Aditi (female, Indian English) and Seoyeon (female, Korean).	November 15, 2017
New feature	Addition of new Speech Marks feature, as well as an expansion of SSML capabilities..	April 19, 2017
New guide	This is the first release of the Amazon Polly Developer Guide.	November 30, 2016