Amazon Polly

Developer Guide

Amazon Polly: Developer Guide

Table of Contents

What Is Amazon Polly? 1
    Are You a First-time User of Amazon Polly? 2
How It Works 3
    What's Next? 3
Getting Started 4
    Step 1: Set Up an Account & User 4
        Step 1.1: Sign up for AWS 4
        Step 1.2: Create an IAM User 5
        Next Step 5
    Step 2: Getting Started Using the Console 6
        Exercise 1: Synthesizing Speech Quick Start (Console) 6
        Exercise 2: Synthesizing Speech (Plain Text Input) 6
        Next Step 7
    Step 3: Getting Started Using the AWS CLI 7
        Step 3.1: Set Up the AWS CLI 7
        Step 3.2: Getting Started Exercise 8
    What's Next? 10
        Set Up Python and Test an Example 10
Using SSML 12
    Using SSML with the Console 12
        Using SSML with the Amazon Polly Console 12
        Next Step 13
    Using SSML with the AWS CLI 13
        Example 1: Passing SSML Through the synthesize-speech Command 13
        Example 2: Synthesizing a Full SSML Document 14
        Example 3: Using Common SSML Tags 14
        Example 4: Controlling Pronunciation 16
Managing Lexicons 18
    Applying Multiple Lexicons 18
    Managing Lexicons Using the Console 19
        Uploading Lexicons Using the Console 20
        Applying Lexicons Using the Console (Synthesize Speech) 20
        Filtering the Lexicon List Using the Console 21
        Downloading Lexicons Using the Console 22
        Deleting a Lexicon Using the Console 22
    Managing Lexicons Using the AWS CLI 22
        PutLexicon 23
        GetLexicon 28
        ListLexicons 30
        DeleteLexicon 31
Example Applications 33
    Python Example 34
        Python Example: index.html 35
        Python Example: server.py 38
    Android Example 44
    iOS Example 46
Limits 48
    Supported Regions 48
    Throttling 48
    Pronunciation Lexicons 49
    SynthesizeSpeech API Operation 49
    Speech Synthesis Markup Language (SSML) 49
CloudTrail Integration 50
    Amazon Polly Information in CloudTrail 50
    Understanding Amazon Polly Log File Entries 51

CloudWatch Integration 53
    Getting CloudWatch Metrics (Console) 53
    Getting CloudWatch Metrics (CLI) 53
    Amazon Polly Metrics 54
    Dimensions for Amazon Polly Metrics 55
API Reference 56
    Actions 56
        DeleteLexicon 57
        DescribeVoices 58
        GetLexicon 60
        ListLexicons 62
        PutLexicon 64
        SynthesizeSpeech 66
    Data Types 68
        Lexicon 69
        LexiconAttributes 70
        LexiconDescription 71
        Voice 72
Authentication and Access Control 73
    Authentication 73
    Access Control 74
    Overview of Managing Access 75
        Amazon Polly Resources and Operations 75
        Understanding Resource Ownership 75
        Managing Access to Resources 76
        Specifying Policy Elements: Actions, Effects, and Principals 77
        Specifying Conditions in a Policy 78
    Using Identity-Based Policies (IAM Policies) 78
        Permissions Required to Use the Amazon Polly Console 79
        AWS Managed (Predefined) Policies for Amazon Polly 79
        Customer Managed Policy Examples 80
    Amazon Polly API Permissions Reference 82
Document History 84
AWS Glossary 85

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, so you can build speech-enabled applications that work in multiple locations and use the ideal voice for your customers. With Amazon Polly, you only pay for the text you synthesize. You can also cache and replay Amazon Polly’s generated speech at no additional cost.

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 not certified for use with regulated workloads such as Payment Card Industry (PCI) Data Security Standard (DSS), HIPAA (Health Insurance Portability and Accountability Act of 1996), or FedRAMP.

Some of the benefits of using Amazon Polly include:

- **High quality** – Amazon Polly uses best-in-class Text-to-Speech (TTS) technology to synthesize natural speech with high pronunciation accuracy (including abbreviations, acronym expansions, date/time interpretations, and homograph disambiguation).

- **Low latency** – Amazon Polly ensures fast response times, which make it a viable option for low-latency use cases such as dialog systems.

- **Support for a large portfolio of languages and voices** – Amazon Polly supports 47 voices and 24 languages, offering male and female options for most languages.

- **Cost-effective** – Amazon Polly's pay-per-use model means there are no setup costs. You can start small and scale up as your application grows.

- **Cloud-based solution** – On-device Text-to-Speech 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, smart phones, etc. In contrast, Text-to-Speech conversion done in the cloud dramatically reduces local resource requirements. This enables support of all the available languages and voices at the best possible quality. Moreover, speech improvements are instantly available to all end-users and do not require additional updates for devices.

Are You a First-time User of Amazon Polly?

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

1. **Amazon Polly: How It Works (p. 3)** – This section introduces various Amazon Polly inputs and options that you can work with in order to create an end-to-end experience.
2. **Getting Started with Amazon Polly (p. 4)** – In this section, you set up your account and test Amazon Polly speech synthesis.
3. **Example Applications (p. 33)** – This section provides additional examples that you can use to explore Amazon Polly.

Amazon Polly: How It Works

Amazon Polly converts input text into life-like speech. You just need to call the `SynthesizeSpeech` method, provide the text you wish to synthesize, select one of the available Text-to-Speech (TTS) voices, and specify an audio output format. Amazon Polly then synthesizes the provided text into a high-quality speech audio stream.

- **Input text** – Provide the text you want to synthesize, and Amazon Polly returns an audio stream. You can provide the input as plain text 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 Using SSML (p. 12).

- **Available voices** – Amazon Polly provides a portfolio of multiple languages and a variety of voices. For most languages you can select from several different voices, including both male and female. You only need to specify the voice name when calling the `SynthesizeSpeech` operation, and then the service uses this voice to convert the text to speech. Amazon Polly is not a translation service —the synthesized speech is in the language of the text. Numbers using digits (for example, *53*, not *fifty-three*) are synthesized in the language of the voice.

- **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 to consume in web and mobile applications. Or, you might request the PCM output format for AWS IoT devices and telephony solutions.

What's Next?

If you are new to Amazon Polly, we recommend that you to read the following topics in order:

- Getting Started with Amazon Polly (p. 4)
- Example Applications (p. 33)
- Limits in Amazon Polly (p. 48)

Getting Started with Amazon Polly

Amazon Polly provides simple API operations that you can easily integrate with your existing applications. For a list of supported operations, see Actions (p. 56). 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 access any of Amazon Polly functionality without having to write any code.

The following sections describe how to get set up and provide an introductory exercise.

Topics

- Step 1: Set Up an AWS Account and Create a User (p. 4)
- Step 2: Getting Started Using the Console (p. 6)
- Step 3: Getting Started Using the AWS CLI (p. 7)
- What's Next? (p. 10)

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

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

1. Step 1.1: Sign up for AWS (p. 4)
2. Step 1.2: Create an IAM User (p. 5)

Step 1.1: 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 are charged only for the services that you use.

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

If you already have an AWS account, skip to the next step. If you don't have an AWS account, perform the steps in the following procedure to create one.

To create an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.
2. Follow the online instructions.

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

Note your AWS account ID because you'll need it for the next step.

Step 1.2: Create an IAM User

Services in AWS, such as Amazon Polly, require that you provide credentials when you access them so that the service can determine whether you have permissions to access the resources owned by that service. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API. However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you use AWS Identity and Access Management (IAM). Create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials.

If you signed up for AWS, but you haven't created an IAM user for yourself, you can create one using the IAM console.

The Getting Started exercises in this guide assume that you have a user (`adminuser`) with administrator privileges. Follow the procedure to create `adminuser` in your account.

To create an administrator user and sign in to the console

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

   Important
   The Getting Started exercises use the adminuser credentials. For added security, when building and testing production application we recommend you create a service-specific administrator user who has permissions for only the Amazon Polly actions. For an example policy that grants Amazon Polly specific permissions, see Example 1: Allow All Amazon Polly Actions (p. 80).

For more information about IAM, see the following:

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

Next Step

Step 2: Getting Started Using the Console (p. 6)

Step 2: Getting Started Using the Console

The Amazon Polly console is the easiest way to get started testing and using Amazon Polly's speech synthesizing. The Amazon Polly console supports synthesizing speech from either plain text or SSML input.

Topics

- Exercise 1: Synthesizing Speech Quick Start (Console) (p. 6)
- Exercise 2: Synthesizing Speech (Plain Text Input) (p. 6)
- Next Step (p. 7)

Exercise 1: Synthesizing Speech Quick Start (Console)

The Quick Start walks you through the fastest way to test the Amazon Polly speech synthesis for speech quality. When you select the **Text-to-Speech** tab, the text field for entering your text is pre-loaded with example text so you can quickly try out Amazon Polly.

To quickly test Amazon Polly

1. Sign in to the AWS Management Console and open the Amazon Polly console at https://console.aws.amazon.com/polly/.
2. Choose the **Text-to-Speech** tab.
3. (Optional) Choose **SSML**.
4. Choose a language and region, then choose a voice.
5. Choose **Listen to speech**.

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

- Exercise 2: Synthesizing Speech (Plain Text Input) (p. 6)
- Using SSML with the Amazon Polly Console (p. 12)
- Applying Lexicons Using the Console (Synthesize Speech) (p. 20)

Exercise 2: Synthesizing Speech (Plain Text Input)

The following procedure synthesizes speech using plain text input. Note how "W3C" and the date "10/3" (October 3rd) are synthesized.

To synthesize speech using plain text input

1. After logging on to the Amazon Polly console, choose **Get started**, and then choose the **Text-to-Speech** tab.
2. Choose the **Plain text** tab.
3. 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.
```

4. For **Choose a language and region**, choose **English US**, then choose the voice you want to use for this text.
5. To listen to the speech immediately, choose **Listen to speech**.
6. To save the speech to a file, do one of the following:

   a. Choose **Save speech to MP3**.
   b. To change to a different file format, choose **Change file format**, choose the file format you want, and then choose **Change**.

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

- Applying Lexicons Using the Console (Synthesize Speech) (p. 20)
- Using SSML with the Amazon Polly Console (p. 12)

Next Step

Step 3: Getting Started Using the AWS CLI (p. 7)

Step 3: Getting Started Using the AWS CLI

Using the AWS CLI you can perform almost all Amazon Polly operations you can perform using the Amazon Polly console. You cannot listen to the synthesized speech using the AWS CLI. Instead, you must save it to a file and then open the file in an application that can play the file.

Topics

- Step 3.1: Set Up the AWS Command Line Interface (AWS CLI) (p. 7)
- Step 3.2: Getting Started Exercise Using the AWS CLI (p. 8)

Step 3.1: Set Up the AWS Command Line Interface (AWS CLI)

Follow the steps to download and configure the AWS Command Line Interface (AWS CLI).

Important
You don't need the AWS CLI to perform the steps in this Getting Started exercise. However, some of the exercises in this guide use the AWS CLI. You can skip this step and go to Step 3.2: Getting Started Exercise Using the AWS CLI (p. 8), and then set up the AWS CLI later when you need it.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:

   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface
2. Add a named profile for the administrator user in the AWS CLI config file. You use this profile when executing the AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*.

```
[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*.

If you specify one of the Amazon Polly supported regions when you configure the AWS CLI, you can omit the following line from the AWS CLI code examples. If you specify a region not supported by Amazon Polly in your AWS CLI configuration (for example, if you're an existing AWS customer using other services in regions that don't support Amazon Polly), you must include the following line:

```
--region polly-supported-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.

Next Step

Step 3.2: Getting Started Exercise Using the AWS CLI (p. 8)

Step 3.2: Getting Started Exercise Using the AWS CLI

Now you can test the speech synthesis offered by Amazon Polly. In this exercise, you call the `SynthesizeSpeech` operation by passing in sample text. You can save the resulting audio as a file and verify its content.

If you specified one of the Amazon Polly supported regions when you configured the AWS CLI, you can omit the following line from the AWS CLI code examples. If you specified a region not supported by Amazon Polly in your AWS CLI configuration (for example, if you're an existing AWS customer using other services in regions that don't support Amazon Polly), you must include the following line:

```
--region polly-supported-aws-region
```

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 OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

```
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 for the synthesis, the voice to use (by providing a voice ID, explained in the following step 3), and the output format. The command saves the resulting audio to the `hello.mp3` file.

In addition to the MP3 file, the above operation produces the following output to the console.

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

2. Play the resulting `hello.mp3` file to verify the synthesized speech.
3. You can get the list of available voices by using the `DescribeVoices` operation. Run the following `describe-voices` AWS CLI command.

```
aws polly describe-voices
```

In response, Amazon Polly returns the list of all available voices. For each voice the response provides the following metadata: voice ID, language code, language name, and the gender of the voice. The following is a sample response:

```
{
    "Voices": [
        {
            "Gender": "Female",
            "Name": "Salli",
            "LanguageName": "US English",
            "Id": "Salli",
            "LanguageCode": "en-US"
        },
        {
            "Gender": "Female",
            "Name": "Joanna",
            "LanguageName": "US English",
            "Id": "Kendra",
            "LanguageCode": "en-US"
        }
    ]
}
```

Optionally, you can specify the language code to find the available voices for a specific language. Amazon Polly supports 47 voices. The following example lists all the voices for Brazilian Portuguese.

```
aws polly describe-voices \
    --language-code pt-BR
```

For a list of language codes, see DescribeVoices (p. 58). These language codes are W3C language identification tags (*`ISO 639 code for the language name`*-*`ISO 3166 country code`*). For example, en-US (US English), en-GB (British English), and es-ES (Spanish), etc.

You can also use the `help` option in the AWS CLI to get the list of language codes:

```
aws polly describe-voices help
```

What's Next?

This guide provides additional examples, some of which are Python code examples that use AWS SDK for Python (Boto) to make API calls to Amazon Polly. We recommend you to set up Python and test the example code provided in the following section. For additional examples, see Example Applications (p. 33).

Set Up Python and Test an Example

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

To test Example Python Code

The following Python code example does the following:

- Uses the AWS SDK for Python (Boto) to send a `SynthesizeSpeech` request to Amazon Polly (by providing simple text as input).
- Accesses the resulting audio stream in the response and saves the audio to a file on your local disk (`speech.mp3`).
- 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 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(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 Mac and Linux. (Darwin = mac, xdg-open = linux).
    opener = "open" if sys.platform == "darwin" else "xdg-open"
subprocess.call([opener, output])
```

For additional examples including an example application, see Example Applications (p. 33).

Using SSML

Amazon Polly generates speech from both plain text input and Speech Synthesis Markup Language (SSML) documents that conform to SSML version 1.1. Using SSML tags, you can customize and control aspects of speech such as pronunciation, volume, and speech rate.

Amazon Polly supports SSML 1.1 as defined in the following W3C recommendation:

- Speech Synthesis Markup Language (SSML) Version 1.1, W3C Recommendation 7 September 2010

Some of the elements in the SSML W3C recommendation are not supported. For more information, see Limits in Amazon Polly (p. 48).

This section provides simple examples of SSML that can be used to generate and control speech output. The examples also provide the `synthesize-speech` AWS CLI command to test these examples.

Using SSML with the Amazon Polly Console

Amazon Polly supports version 1.1 SSML as defined by the W3C. This section covers how to use SSML input for speech synthesis in the Amazon Polly console.

Using SSML with the Amazon Polly Console

The following procedure synthesizes speech using SSML input. Except for steps 3 and 4 below, the steps in this example are identical to those in Exercise 2: Synthesizing Speech (Plain Text Input) (p. 6).

To synthesize speech using the Amazon Polly console (SSML input)

In this example we introduce SSML tagging to substitute "World Wide Web Consortium" for "W3C". Compare the results of this exercise with that of Applying Lexicons Using the Console (Synthesize Speech) (p. 20) for both US English and another language.

1. Sign in to the AWS Management Console and open the Amazon Polly console at https://console.aws.amazon.com/polly/.
2. Choose **Get started** and then, if needed, the **Text-to-Speech** tab.
3. Choose the **SSML** tab.
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, "Score!" quite loudly.</speak>
```

The SSML tags inform Amazon Polly that the text should be rendered in a specified way.

- `<break time="1s"/>` instructs Amazon Polly to pause 1 second between the sentences.
- `<sub alias="World Wide Web Consortium">W3C</sub>` instructs Amazon Polly to substitute "World Wide Web Consortium" for the acronym "W3C".

For more information on SSML with examples, see Using SSML with the AWS CLI (p. 13)

5. For **Choose a language and region**, choose **English US**, then choose the voice you want.
6. To listen to the speech immediately, choose **Listen to speech**.
7. To save the speech to a file, choose **Save speech to MP3**.

   To change to a different file format, choose **Change file format**, choose the file format you want, and then choose **Change**.

Related Console Examples

- Exercise 2: Synthesizing Speech (Plain Text Input) (p. 6)
- Applying Lexicons Using the Console (Synthesize Speech) (p. 20)

Next Step

Applying Lexicons Using the Console (Synthesize Speech) (p. 20)

Using SSML with the AWS CLI

The following topics demonstrate how to use SSML input with the AWS CLI for Amazon Polly.

Example 1: Passing SSML Through the synthesize-speech Command

In the following `synthesize-speech` command, you specify a simple SSML string with only the required opening and closing `<speak></speak>` tags (optionally, you can specify the full document header too). Because plain text is the default, the command also specifies the `--text-type` parameter to indicate that the input text is SSML. The only required elements for an SSML string are the input text (targeted by the generated speech), `output-format`, and `voice-id`.

The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

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

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

Example 2: Synthesizing a Full SSML Document

In this example you save SSML content to a file and specify the file name in the `synthesize-speech` command. This example uses the following SSML.

```
<?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>
```

Note that 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 voice selected affect the `SynthesizeSpeech` operation, see Using the xml:lang Attribute of the <speak> Element (p. 17).

To test the SSML

1. Save the SSML to a file (`example.xml`).
2. Run the following `synthesize-speech` command from the path where the XML file is stored and specify the SSML as input.

   The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

   ```
   aws polly synthesize-speech \
   --text-type ssml \
   --text file://example.xml \
   --output-format mp3 \
   --voice-id Joanna \
   speech.mp3
   ```
3. Play the `speech.mp3` file to verify the synthesized speech.

Example 3: Using Common SSML Tags

This section explains how to use some common SSML tags to achieve specific results. For more examples, see Speech Synthesis Markup Language (SSML) Version 1.1.

You can use the `synthesize-speech` command to test the examples in this section.

Using the <break> Element

The following `synthesize-speech` command uses the `<break>` element to add a 300 millisecond delay between the words "Hello" and "World" in the resulting speech.

```
"<speak>Hello <break time='300ms'/> World</speak>"
```

The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

```
aws polly synthesize-speech \
--text-type ssml \
--text "<speak>Hello <break time='300ms'/> World</speak>" \
--output-format mp3 \
--voice-id Joanna \
speech.mp3
```

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

Using the <prosody> Element

This element enables you to control pitch, speaking rate, and volume of speech.

- The following SSML uses the <`prosody`> element to control volume:

  ```
  "<speak><prosody volume='+20dB'>Hello world</prosody></speak>"
  ```

  The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

  ```
  aws polly synthesize-speech \
  --text-type ssml \
  --text "<speak><prosody volume='+20dB'>Hello world</prosody></speak>" \
  --output-format mp3 \
  --voice-id Joanna \
  speech.mp3
  ```
- The following SSML uses the <`prosody`> element to control pitch:

  ```
  "<speak><prosody pitch='x-high'>Hello world</prosody></speak>"
  ```

  The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

  ```
  aws polly synthesize-speech \
  --text-type ssml \
  --text "<speak><prosody pitch='x-high'>Hello world</prosody></speak>" \
  --output-format mp3 \
  --voice-id Joanna \
  speech.mp3
  ```
- The following SSML uses the <`prosody`> element to specify the speech rate:

  ```
  "<speak><prosody rate='x-fast'>Hello world</prosody></speak>"
  ```

  The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

  ```
  aws polly synthesize-speech \
  --text-type ssml \
  --text "<speak><prosody rate='x-fast'>Hello world</prosody></speak>" \
  ```

```
--output-format mp3 \
--voice-id Joanna \
speech.mp3
```

- You can specify multiple attributes in a `<prosody>` element, as shown in the following example:

  ```
  "<speak><prosody volume='x-loud' pitch='x-high' rate='x-fast'>Hello world</
  prosody></speak>"
  ```

  The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

  ```
  aws polly synthesize-speech \
  --text-type ssml \
  --text "<speak><prosody volume='x-loud' pitch='x-high' rate='x-fast'>Hello
   world</prosody></speak>" \
  --output-format mp3 \
  --voice-id Joanna \
  speech.mp3
  ```

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

Using the <emphasis> Element

This element enables you to specify the stress or prominence to apply when speaking a specified word or phrase.

```
"<speak><emphasis level='strong'>Hello</emphasis> world how are you?</speak>"
```

The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

```
aws polly synthesize-speech \
--text-type ssml \
--text "<speak><emphasis level='strong'>Hello</emphasis> world how are you?</
speak>" \
--output-format mp3 \
--voice-id Joanna \
speech.mp3
```

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

Example 4: Controlling Pronunciation

This example explains some of the common SSML tags you can use to control pronunciation.

Using the <say-as> Element

This element enables you to provide information about the type of text contained within the element. In the following SSML, `<say-as>` indicates that the text 4/6 should be treated as a date value with format day/month.

```
"<speak>Today is <say-as interpret-as='date' format='dm' >4/6</say-as></
speak>"
```

The following AWS CLI example is formatted for Unix, Linux, and OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

```
aws polly synthesize-speech \
--text-type ssml \
--text "<speak>Today is <say-as interpret-as='date' format='dm' >4/6</say-
as></speak>" \
--output-format mp3 \
--voice-id Joanna \
speech.mp3
```

The resulting speech says "Today is June 4th". The `<say-as>` tag describes how the text should be interpreted by providing additional context via the `interpret-as` attribute.

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

Using the xml:lang Attribute of the <speak> Element

You can improve pronunciation of words that are foreign to the input text language by specifying the target language using the `xml:lang` attribute of the `<speak>` element. This forces the TTS engine to apply different pronunciation rules for the words that are specific to the target language. The examples below show different combinations of languages for the input text and the voices you can specify in the `synthesize-speech` call. You can test these examples using the `synthesize-speech` command to verify the results.

In this example, the chosen voice is a US English voice. Amazon Polly assumes the input text is in the same language as the selected voice. To achieve a Spanish pronunciation of specific words, you need to mark the targeted words as Spanish.

```
aws polly synthesize-speech \
--text-type ssml \
--text "<speak>That restaurant is terrific. <xml:lang=\"es-ES\">Mucho gusto</
xml:lang>!</speak>" \
--output-format mp3 \
--voice-id Joanna \
speech.mp3
```

Because the language of the input text is specified, Amazon Polly maps the resulting Spanish phonemes to English phonemes of the smallest acoustic distance. As a result, Salli reads the text as a US native speaker who knows the correct Spanish pronunciation, but in a US English accent.

Note
This practice is limited to pairs of languages available in the Amazon Polly language portfolio. Some language pairs work better than others because of the phonological structure of the languages.

Play the resulting `speech.mp3` file to listen to the synthesized speech.

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 (p. 66).

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

- Applying Multiple Lexicons (p. 18)
- Managing Lexicons Using the Amazon Polly Console (p. 19)
- Managing Lexicons Using the AWS CLI (p. 22)

Applying 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 Using the Console (Synthesize Speech) (p. 20).

Managing Lexicons Using the Amazon Polly Console

You can use the Amazon Polly console to upload, download, apply, filter, and delete lexicons. The following procedures demonstrate each of these processes.

Uploading Lexicons Using the Console

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://console.aws.amazon.com/polly/.
2. Choose the **Lexicons** tab.
3. Choose **Upload**.
4. Browse to find the lexicon that you want to upload. You can use only PLS files that use the .pls and .xml extensions.
5. Choose **Open**. If a lexicon by the same name (whether a .pls or .xml file) already exists, uploading the lexicon will overwrite the existing lexicon.

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://console.aws.amazon.com/polly/.
2. Choose the **Text-to-Speech** tab.
3. Choose **Customize pronunciation of words or phrases using lexicons**, then choose **Upload lexicon**.
4. Browse to find the lexicon that you want to upload. You can use only PLS files that use the .pls and .xml extensions.
5. Choose **Open**. If a lexicon with the same name (whether a .pls or .xml file) already exists, uploading the lexicon will overwrite the existing lexicon.

Applying Lexicons Using the Console (Synthesize Speech)

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 Using the PutLexicon Operation (p. 23). 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 lexicons to substitute "World Wide Web Consortium" for "W3C". Compare the results of this exercise with that of Using SSML with the Amazon Polly Console (p. 12) for both US English and another language.

1. Sign in to the AWS Management Console and open the Amazon Polly console at https://console.aws.amazon.com/polly/.
2. Do one of the following:
   - Choose the **Plain text** tab 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.
     ```
   - Choose the **SSML** tab 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 **Choose a language and region** list, choose English US, then choose a voice you want to use for this text.
4. Choose **Customize pronunciation of words or phrases using lexicons**.
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.
6. To listen to the speech immediately, choose **Listen to speech**.
7. To save the speech to a file,
   a. Choose **Save speech to MP3**.
   b. To change to a different file format, choose **Change file format**, choose the file format you want, and then choose **Change**.

Repeat the previous steps, but choose a different language and notice the difference in the output.

Filtering the Lexicon List Using the Console

The following procedure describes how to filter the lexicons list so that only lexicons of a chosen language are displayed.

To filter the lexicons listed by language

1. Sign in to the AWS Management Console and open the Amazon Polly console at https://console.aws.amazon.com/polly/.
2. Choose the **Lexicons** tab.

3. Choose **Filter**.
4. From the list of languages, choose the language you want to filter on.

   The list displays only the lexicons for the chosen language.

Downloading Lexicons Using 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.

To download one or more lexicons

1. Sign in to the AWS Management Console and open the Amazon Polly console at https://console.aws.amazon.com/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**.

Deleting a Lexicon Using the Console

To delete 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.

1. Sign in to the AWS Management Console and open the Amazon Polly console at https://console.aws.amazon.com/polly/.
2. Choose the **Lexicons** tab.
3. Choose one or more lexicons that you want to delete from the list.
4. Choose **Delete**.
5. Choose **Delete** to remove the lexicon from the region or **Cancel** to keep it.

Managing Lexicons Using the AWS CLI

The following topics cover the AWS CLI commands needed to manage your pronunciation lexicons.

Topics

- Using the PutLexicon Operation (p. 23)
- Using the GetLexicon Operation (p. 28)
- Using the ListLexicons Operations (p. 30)
- Using the DeleteLexicon Operation (p. 31)

Using the PutLexicon Operation

With Amazon Polly, you can use PutLexicon (p. 64) 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 (p. 66) request that you want to apply before the service starts synthesizing the text. For more information, see Managing Lexicons (p. 18).

This section provides example lexicons and step-by-step instructions for storing and testing them.

> **Note**
> These lexicons 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.

Example 1: Lexicon with One Lexeme

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-1 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 word "pecan" (for example, "Pecan pie is delicious").

Example 2: Lexicon with Multiple Lexemes

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 WWW Consortium. The revised text appears as follows:

  ```
  The WWW 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-1 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.

Example 3: Specifying Multiple Lexicons

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
   ```

Example 4: Python Code for PutLexicon

The following Python code example uses the AWS SDK for Python (Boto) to store a lexicon. 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 Step 3.1: Set Up the AWS Command Line Interface (AWS CLI) (p. 7).

```
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))
```

Using the GetLexicon Operation

Amazon Polly provides the GetLexicon (p. 60) 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 Using the PutLexicon Operation (p. 23).

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-1:account-id:lexicon/example",
        "Size": 495
    }
}
```

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 Step 3.1: Set Up the AWS Command Line Interface (AWS CLI) (p. 7).

```
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.")
```

Using the ListLexicons Operations

Amazon Polly provides the ListLexicons (p. 62) 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-1 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 Python code example uses the AWS SDK for Python (Boto) 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 Step 3.1: Set Up the AWS Command Line Interface (AWS CLI) (p. 7).

```
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))
```

Using the DeleteLexicon Operation

Amazon Polly provides the DeleteLexicon (p. 57) 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 OS X. To run this sample at a Windows command prompt, replace the backslash (\) Unix continuation character at the end of each line with the carrot (^) Windows continuation character.

```
aws polly delete-lexicon \
--name example
```

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 Step 3.1: Set Up the AWS Command Line Interface (AWS CLI) (p. 7).

```
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.")
```

Example Applications

This section provides additional examples that you can use to explore Amazon Polly. Before you start using these examples, we recommend that you first read Amazon Polly: How It Works (p. 3) and follow the steps described in Getting Started with Amazon Polly (p. 4).

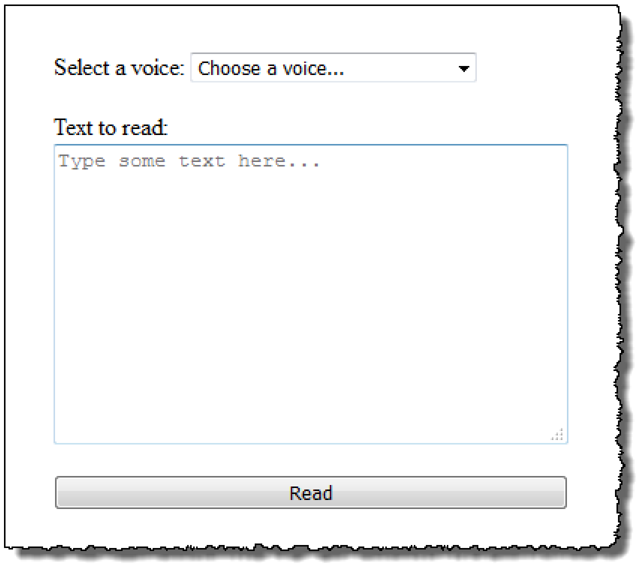
Topics

- Python Example (HTML5 Client and Python Server) (p. 34)
- Android Example (p. 44)
- iOS Example (p. 46)

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):



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) (p. 38).
2. Save the HTML5 client code as `index.html`. For the code, see Python Example: HTML5 User Interface (index.html) (p. 35).
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 (Ctrl+Z on Windows) 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) (p. 34).

```
<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'
        };

        /**
         * 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) (p. 34).

```
"""
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"}
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)
            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()
```

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 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();
    }
});
```

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 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.continue(successBlock: { (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
})
```

Limits in Amazon Polly

The following are limits to be aware of when using Amazon Polly.

Supported Regions

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

Throttling

- Throttle rate per IP address: 20 requests per second (rps) with a burst limit of 40 rps.
- Throttle rate per account: 10 requests per second (rps) with a burst limit of 20 rps.

  Concurrent connections per account: 30
- Throttle rate per operation:

Operation	**Limit**
Lexicon	

`DeleteLexicon`
`PutLexicon`
`GetLexicon`
`ListLexicons`
	Any 2 requests per second (rps) from these operations combined.
	
	Maximum allowed burst of 4 rps.

Speech	
`DescribeVoices`	10 rps with a burst limit of 20 rps
`SynthesizeSpeech`	10 rps with a burst limit of 20 rps

Throttling limits may be increased by contacting customer service. For more information, see AWS Service Limits.

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 4,000 characters in size.
- You can specify up to 100 characters for each <phoneme> or <alias> replacement in a lexicon.

For information about using lexicons, see Managing Lexicons (p. 18).

SynthesizeSpeech API Operation

Note the following limits related to using the SynthesizeSpeech API operation:

- The size of the input text can be up to 1500 billed characters (3000 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 5 minutes, after which, any remaining speech is cut off.

For more information, see SynthesizeSpeech (p. 66).

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 Using SSML (p. 12).

Logging Amazon Polly API Calls with AWS CloudTrail

Amazon Polly is integrated with CloudTrail, a service that captures all of the Amazon Polly API calls and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the Amazon Polly console or from your code to the Amazon Polly APIs. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Polly, the source IP address from which the request was made, who made the request, when it was made, and so on.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon Polly Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to Amazon Polly actions are tracked in CloudTrail log files, where they are written with other AWS service records. CloudTrail determines when to create and write to a new file based on a time period and file size.

All Amazon Polly actions are logged by CloudTrail and are documented in the Amazon Polly API Reference (p. 56). The following actions are supported.

- DeleteLexicon (p. 57)
- DescribeVoices (p. 58)
- GetLexicon (p. 60)
- ListLexicons (p. 62)
- PutLexicon (p. 64)
- SynthesizeSpeech (p. 66)

Every log entry contains information about who generated the request. The user identity information in the log entry helps you determine the following:

- Whether the request was made with root or 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.

You can store your log files in your Amazon S3 bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

If you want to be notified upon log file delivery, you can configure CloudTrail to publish Amazon SNS notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

You can also aggregate Amazon Polly log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket.

For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

Understanding Amazon Polly Log File Entries

CloudTrail log files can contain one or more log entries. Each entry lists multiple JSON-formatted events. A log entry 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. Log entries are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

Because of potential confidentiality issues, log entries do not contain the synthesized text. Instead, this text is redacted in the log entry.

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

```
{
    "Records": [
        {
            "awsRegion": "us-east-1",
            "eventID": "19bd70f7-5e60-4cdc-9825-936c552278ae",
            "eventName": "SynthesizeSpeech",
            "eventSource": "tts.amazonaws.com",
            "eventTime": "2016-11-02T03:49:39Z",
            "eventType": "AwsApiCall",
            "eventVersion": "1.05",
            "recipientAccountId": "123456789012",
            "requestID": "414288c2-a1af-11e6-b17f-d7cfc06cb461",
            "requestParameters": {
                "lexiconNames": [
                    "SampleLexicon"
                ],
                "outputFormat": "mp3",
                "sampleRate": "22050",
                "text": "**********",
                "textType": "text",
                "voiceId": "Kendra"
            },
            "responseElements": {
                "contentType": "audio/mpeg",
                "requestCharacters": 25
            },
            "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"
            }
        }

    ]
}
```

The `eventName` element identifies the action that occurred and may include date and version information, such as `"SynthesizeSpeech20161128"`, nevertheless it is still referring to the same public API.

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://console.aws.amazon.com/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 (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 Valid Statistics: Minimum, Maximum, Average, SampleCount Unit: milliseconds
`2XXCount`	HTTP 200 level code returned upon a successful response. Valid Dimensions: Operation Valid Statistics: Average, SampleCount, Sum Unit: Count

Metric	Description
`4XXCount`	HTTP 400 level error code returned upon an error. For each successful response, a zero (0) is emitted. Valid Dimensions: Operation Valid Statistics: Average, SampleCount, Sum Unit: Count
`5XXCount`	HTTP 500 level error code returned upon an error. 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 `SynthesizeSpeech`, `PutLexicon`, `DescribeVoices`, etc.

Amazon Polly API Reference

This section contains the Amazon Polly API reference.

Note
Authenticated API calls must be signed using the Signature Version 4 Signing Process. For more information, see Signing AWS API Requests in the *Amazon Web Services General Reference*.

Topics

- Actions (p. 56)
- Data Types (p. 68)

Actions

The following actions are supported:

- DeleteLexicon (p. 57)
- DescribeVoices (p. 58)
- GetLexicon (p. 60)
- ListLexicons (p. 62)
- PutLexicon (p. 64)
- SynthesizeSpeech (p. 66)

DeleteLexicon

Deletes the specified pronunciation lexicon stored in an AWS Region. A lexicon which has been deleted is not available for speech synthesis, nor is it possible to retrieve it using either the `GetLexicon` or `ListLexicon` APIs.

For more information, see Managing Lexicons.

Request Syntax

```
DELETE /v1/lexicons/LexiconName HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

Name (p. 57)

The name of the lexicon to delete. Must be an existing lexicon in the region.

Pattern: `[0-9A-Za-z]{1,20}`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

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

Errors

LexiconNotFoundException

Amazon Polly can't find the specified lexicon. This could be caused by a lexicon that is missing, its name is misspelled or specifying a lexicon that is in a different region.

Verify that the lexicon exists, is in the region (see ListLexicons (p. 62)) and that you spelled its name is spelled correctly. Then try again.

HTTP Status Code: 404

ServiceFailureException

An unknown condition has caused a service failure.

HTTP Status Code: 500

DescribeVoices

Returns the list of voices that are available for use when requesting speech synthesis. Each voice speaks a specified language, is either male or female, and is identified by an ID, which is the ASCII version of the voice name.

When synthesizing speech (`SynthesizeSpeech`), you provide the voice ID for the voice you want from the list of voices returned by `DescribeVoices`.

For example, you want your news reader application to read news in a specific language, but giving a user the option to choose the voice. Using the `DescribeVoices` operation you can provide the user with a list of available voices to select from.

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.

This operation requires permissions to perform the `polly:DescribeVoices` action.

Request Syntax

```
GET /v1/voices?LanguageCode=LanguageCode&NextToken=NextToken HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

LanguageCode (p. 58)

The language identification tag (ISO 639 code for the language name-ISO 3166 country code) for filtering the list of voices returned. If you don't specify this optional parameter, all available voices are returned.

Valid Values: `cy-GB | da-DK | de-DE | en-AU | en-GB | en-GB-WLS | en-IN | en-US | es-ES | es-US | fr-CA | fr-FR | is-IS | it-IT | ja-JP | nb-NO | nl-NL | pl-PL | pt-BR | pt-PT | ro-RO | ru-RU | sv-SE | tr-TR`

NextToken (p. 58)

An opaque pagination token returned from the previous `DescribeVoices` operation. If present, this indicates where to continue the listing.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "NextToken": "string",
   "Voices": [
      {
         "Gender": "string",
         "Id": "string",
         "LanguageCode": "string",
         "LanguageName": "string",
         "Name": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

NextToken (p. 58)
The pagination token to use in the next request to continue the listing of voices. `NextToken` is returned only if the response is truncated.

Type: String

Voices (p. 58)
A list of voices with their properties.

Type: array of Voice (p. 72) objects

Errors

InvalidNextTokenException
The NextToken is invalid. Verify that it's spelled correctly, and then try again.

HTTP Status Code: 400

ServiceFailureException
An unknown condition has caused a service failure.

HTTP Status Code: 500

GetLexicon

Returns the content of the specified pronunciation lexicon stored in an AWS Region. For more information, see Managing Lexicons.

Request Syntax

```
GET /v1/lexicons/LexiconName HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

Name (p. 60)

Name of the lexicon.

Pattern: `[0-9A-Za-z]{1,20}`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Lexicon": {
      "Content": "string",
      "Name": "string"
   },
   "LexiconAttributes": {
      "Alphabet": "string",
      "LanguageCode": "string",
      "LastModified": number,
      "LexemesCount": number,
      "LexiconArn": "string",
      "Size": number
   }
}
```

Response Elements

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

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

Lexicon (p. 60)

Lexicon object that provides name and the string content of the lexicon.

Type: Lexicon (p. 69) object

LexiconAttributes (p. 60)

Metadata of the lexicon, including phonetic alphabetic used, language code, lexicon ARN, number of lexemes defined in the lexicon, and size of lexicon in bytes.

Type: LexiconAttributes (p. 70) object

Errors

LexiconNotFoundException

Amazon Polly can't find the specified lexicon. This could be caused by a lexicon that is missing, its name is misspelled or specifying a lexicon that is in a different region.

Verify that the lexicon exists, is in the region (see ListLexicons (p. 62)) and that you spelled its name is spelled correctly. Then try again.

HTTP Status Code: 404

ServiceFailureException

An unknown condition has caused a service failure.

HTTP Status Code: 500

ListLexicons

Returns a list of pronunciation lexicons stored in an AWS Region. For more information, see Managing Lexicons.

Request Syntax

```
GET /v1/lexicons?NextToken=NextToken HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

NextToken (p. 62)
An opaque pagination token returned from previous `ListLexicons` operation. If present, indicates where to continue the list of lexicons.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Lexicons": [
      {
         "Attributes": {
            "Alphabet": "string",
            "LanguageCode": "string",
            "LastModified": number,
            "LexemesCount": number,
            "LexiconArn": "string",
            "Size": number
         },
         "Name": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

Lexicons (p. 62)
A list of lexicon names and attributes.
Type: array of LexiconDescription (p. 71) objects

NextToken (p. 62)
The pagination token to use in the next request to continue the listing of lexicons. `NextToken` is returned only if the response is truncated.
Type: String

Errors

InvalidNextTokenException

The NextToken is invalid. Verify that it's spelled correctly, and then try again.

HTTP Status Code: 400

ServiceFailureException

An unknown condition has caused a service failure.

HTTP Status Code: 500

PutLexicon

Stores a pronunciation lexicon in an AWS Region. If a lexicon with the same name already exists in the region, it is overwritten by the new lexicon. Lexicon operations have eventual consistency, therefore, it might take some time before the lexicon is available to the SynthesizeSpeech operation.

For more information, see Managing Lexicons.

Request Syntax

```
PUT /v1/lexicons/LexiconName HTTP/1.1
Content-type: application/json

{
   "Content": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

Name (p. 64)
: Name of the lexicon. The name must follow the regular express format [0-9A-Za-z]{1,20}. That is, the name is a case-sensitive alphanumeric string up to 20 characters long.

  Pattern: `[0-9A-Za-z]{1,20}`

Request Body

The request accepts the following data in JSON format.

Content (p. 64)
: Content of the PLS lexicon as string data.

  Type: String

  Required: Yes

Response Syntax

```
HTTP/1.1 200
```

Response Elements

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

Errors

InvalidLexiconException
: Amazon Polly can't find the specified lexicon. Verify that the lexicon's name is spelled correctly, and then try again.

  HTTP Status Code: 400

LexiconSizeExceededException
: The maximum size of the specified lexicon would be exceeded by this operation.

  HTTP Status Code: 400

MaxLexemeLengthExceededException
: The maximum size of the lexeme would be exceeded by this operation.

HTTP Status Code: 400

MaxLexiconsNumberExceededException
The maximum number of lexicons would be exceeded by this operation.

HTTP Status Code: 400

ServiceFailureException
An unknown condition has caused a service failure.

HTTP Status Code: 500

UnsupportedPlsAlphabetException
The alphabet specified by the lexicon is not a supported alphabet. Valid values are `x-sampa` and `ipa`.

HTTP Status Code: 400

UnsupportedPlsLanguageException
The language specified in the lexicon is unsupported. For a list of supported languages, see Lexicon Attributes.

HTTP Status Code: 400

SynthesizeSpeech

Synthesizes UTF-8 input, plain text or SSML, to a stream of bytes. SSML input must be valid, well-formed SSML. Some alphabets might not be available with all the voices (for example, Cyrillic might not be read at all by English voices) unless phoneme mapping is used. For more information, see How it Works.

Request Syntax

```
POST /v1/speech HTTP/1.1
Content-type: application/json

{
   "LexiconNames": [ "string" ],
   "OutputFormat": "string",
   "SampleRate": "string",
   "Text": "string",
   "TextType": "string",
   "VoiceId": "string"
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

LexiconNames (p. 66)

List of one or more pronunciation lexicon names you want the service to apply during synthesis. Lexicons are applied only if the language of the lexicon is the same as the language of the voice. For information about storing lexicons, see PutLexicon.

Type: array of Strings

Array Members: Maximum number of 5 items.

Pattern: `[0-9A-Za-z]{1,20}`

Required: No

OutputFormat (p. 66)

The audio format in which the resulting stream will be encoded.

Type: String

Valid Values: `mp3 | ogg_vorbis | pcm`

Required: Yes

SampleRate (p. 66)

The audio frequency specified in Hz.

The valid values for `mp3` and `ogg_vorbis` are "8000", "16000", and "22050". The default value is "22050".

Valid values for `pcm` are "8000" and "16000" The default value is "16000".

Type: String

Required: No

Text (p. 66)

Input text to synthesize. If you specify `ssml` as the `TextType`, follow the SSML format for the input text.

Type: String

Required: Yes

TextType (p. 66)

Specifies whether the input text is plain text or SSML. The default value is plain text. For more information, see Using SSML.

Type: String

Valid Values: `ssml | text`

Required: No

VoiceId (p. 66)

Voice ID to use for the synthesis. You can get a list of available voice IDs by calling the DescribeVoices operation.

Type: String

Valid Values: `Geraint | Gwyneth | Mads | Naja | Hans | Marlene | Nicole | Russell | Amy | Brian | Emma | Raveena | Ivy | Joanna | Joey | Justin | Kendra | Kimberly | Salli | Conchita | Enrique | Miguel | Penelope | Chantal | Celine | Mathieu | Dora | Karl | Carla | Giorgio | Mizuki | Liv | Lotte | Ruben | Ewa | Jacek | Jan | Maja | Ricardo | Vitoria | Cristiano | Ines | Carmen | Maxim | Tatyana | Astrid | Filiz`

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-Type: ContentType
x-amzn-RequestCharacters: RequestCharacters

AudioStream
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The response returns the following HTTP headers.

ContentType (p. 67)

Specifies the type audio stream. This should reflect the `OutputFormat` parameter in your request.

- If you request `mp3` as the `OutputFormat`, the `ContentType` returned is audio/mpeg.
- If you request `ogg_vorbis` as the `OutputFormat`, the `ContentType` returned is audio/ogg.
- If you request `pcm` as the `OutputFormat`, the `ContentType` returned is audio/pcm in a signed 16-bit, 1 channel (mono), little-endian format.

RequestCharacters (p. 67)

Number of characters synthesized.

The response returns the following as the HTTP body.

<varlistentry> **AudioStream (p. 67)**

Stream containing the synthesized speech.

</varlistentry>

Errors

InvalidSampleRateException

The specified sample rate is not valid.

HTTP Status Code: 400

InvalidSsmlException

The SSML you provided is invalid. Verify the SSML syntax, spelling of tags and values, and then try again.

HTTP Status Code: 400

LexiconNotFoundException

Amazon Polly can't find the specified lexicon. This could be caused by a lexicon that is missing, its name is misspelled or specifying a lexicon that is in a different region.

Verify that the lexicon exists, is in the region (see ListLexicons (p. 62)) and that you spelled its name is spelled correctly. Then try again.

HTTP Status Code: 404

ServiceFailureException

An unknown condition has caused a service failure.

HTTP Status Code: 500

TextLengthExceededException

The value of the "Text" parameter is longer than the accepted limits. The limit for input text is a maximum of 3000 characters total, of which no more than 1500 can be billed characters. SSML tags are not counted as billed characters.

HTTP Status Code: 400

Data Types

The following data types are supported:

- Lexicon (p. 69)
- LexiconAttributes (p. 70)
- LexiconDescription (p. 71)
- Voice (p. 72)

Lexicon

Provides lexicon name and lexicon content in string format. For more information, see Pronunciation Lexicon Specification (PLS) Version 1.0.

Contents

Content

Lexicon content in string format. The content of a lexicon must be in PLS format.

Type: String

Required: No

Name

Name of the lexicon.

Type: String

Pattern: `[0-9A-Za-z]{1,20}`

Required: No

LexiconAttributes

Contains metadata describing the lexicon such as the number of lexemes, language code, and so on. For more information, see Managing Lexicons.

Contents

Alphabet

Phonetic alphabet used in the lexicon. Valid values are `ipa` and `x-sampa`.

Type: String

Required: No

LanguageCode

Language code that the lexicon applies to. A lexicon with a language code such as "en" would be applied to all English languages (en-GB, en-US, en-AUS, en-WLS, and so on.

Type: String

Valid Values: `cy-GB | da-DK | de-DE | en-AU | en-GB | en-GB-WLS | en-IN | en-US | es-ES | es-US | fr-CA | fr-FR | is-IS | it-IT | ja-JP | nb-NO | nl-NL | pl-PL | pt-BR | pt-PT | ro-RO | ru-RU | sv-SE | tr-TR`

Required: No

LastModified

Date lexicon was last modified (a timestamp value).

Type: Timestamp

Required: No

LexemesCount

Number of lexemes in the lexicon.

Type: Integer

Required: No

LexiconArn

Amazon Resource Name (ARN) of the lexicon.

Type: String

Required: No

Size

Total size of the lexicon, in characters.

Type: Integer

Required: No

LexiconDescription

Describes the content of the lexicon.

Contents

Attributes

Provides lexicon metadata.

Type: LexiconAttributes (p. 70) object

Required: No

Name

Name of the lexicon.

Type: String

Pattern: `[0-9A-Za-z]{1,20}`

Required: No

Voice

Description of the voice.

Contents

Gender

Gender of the voice.

Type: String

Valid Values: `Female | Male`

Required: No

Id

Amazon Polly assigned voice ID. This is the ID that you specify when calling the `SynthesizeSpeech` operation.

Type: String

Valid Values: `Geraint | Gwyneth | Mads | Naja | Hans | Marlene | Nicole | Russell | Amy | Brian | Emma | Raveena | Ivy | Joanna | Joey | Justin | Kendra | Kimberly | Salli | Conchita | Enrique | Miguel | Penelope | Chantal | Celine | Mathieu | Dora | Karl | Carla | Giorgio | Mizuki | Liv | Lotte | Ruben | Ewa | Jacek | Jan | Maja | Ricardo | Vitoria | Cristiano | Ines | Carmen | Maxim | Tatyana | Astrid | Filiz`

Required: No

LanguageCode

Language code of the voice.

Type: String

Valid Values: `cy-GB | da-DK | de-DE | en-AU | en-GB | en-GB-WLS | en-IN | en-US | es-ES | es-US | fr-CA | fr-FR | is-IS | it-IT | ja-JP | nb-NO | nl-NL | pl-PL | pt-BR | pt-PT | ro-RO | ru-RU | sv-SE | tr-TR`

Required: No

LanguageName

Human readable name of the language in English.

Type: String

Required: No

Name

Name of the voice (for example, Salli, Kendra, etc.). This provides a human readable voice name that you might display in your application.

Type: String

Required: No

Authentication and Access Control for Amazon Polly

Access to Amazon Polly requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon Polly lexicon or an Amazon Elastic Compute Cloud (Amazon EC2) instance. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon Polly to help secure access to your resources.

- Authentication (p. 73)
- Access Control (p. 74)

Authentication

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

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials* and they provide complete access to all of your AWS resources.

  Important
  For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the *IAM User Guide*.
- **IAM user** – An IAM user is simply an identity within your AWS account that has specific custom permissions (for example, permissions to create a lexicon in Amazon Polly). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. Amazon Polly supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

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

  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **Applications running on Amazon EC2** – Instead of storing access keys within the EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 in the *IAM User Guide*.

Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon Polly resources. For example, you must have permissions to create an Amazon Polly lexicon.

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

- Overview of Managing Access Permissions to Your Amazon Polly Resources (p. 75)

- Using Identity-Based Policies (IAM Policies) for Amazon Polly (p. 78)
- Amazon Polly API Permissions: Actions, Permissions, and Resources Reference (p. 82)

Overview of Managing Access Permissions to Your Amazon Polly Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

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

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

- Amazon Polly Resources and Operations (p. 75)
- Understanding Resource Ownership (p. 75)
- Managing Access to Resources (p. 76)
- Specifying Policy Elements: Actions, Effects, and Principals (p. 77)
- Specifying Conditions in a Policy (p. 78)

Amazon Polly Resources and Operations

In Amazon Polly, the primary resource is *a lexicon*. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.

These resources and subresources have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

Resource Type	**ARN Format**
Lexicon	`arn:aws:polly:`*`region`*`:`*`account-id`*`:lexicon/`*`LexiconName`*

Amazon Polly provides a set of operations to work with Amazon Polly resources. For a list of available operations, see Amazon Polly Amazon Polly API Reference (p. 56).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a lexicon, your AWS account is the owner of the resource (in Amazon Polly, the resource is a lexicon).

- If you create an IAM user in your AWS account and grant permissions to create a lexicon to that user, the user can create a lexicon. However, your AWS account, to which the user belongs, owns the lexicon resource.
- If you create an IAM role in your AWS account with permissions to create a lexicon, anyone who can assume the role can create a lexicon. Your AWS account, to which the user belongs, owns the lexicon resource.

Managing Access to Resources

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

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

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

Topics

- Identity-Based Policies (IAM Policies) (p. 76)
- Resource-Based Policies (p. 77)

Identity-Based Policies (IAM Policies)

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

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create a Amazon Polly resource, such as a lexicon, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – 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. Therefore, the `Resource` value is indicated by the ARN. For example: `arn:aws:polly:`*`us-east-1`*`:`*`account-`*

id:lexicon/* as the Resource value specifies permissions on all owned lexicons within the us-east-1 region.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowPut-Get-ListActions",
      "Effect": "Allow",
      "Action": [
         "polly:PutLexicon",
         "polly:GetLexicon",
         "polly:ListLexicons"],
      "Resource": "arn:aws:polly:us-east-1:account-id:lexicon/*"
      }
   ]
}
```

For more information about using identity-based policies with Amazon Polly, see Using Identity-Based Policies (IAM Policies) for Amazon Polly (p. 78). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

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

Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Polly resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon Polly defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see Amazon Polly Resources and Operations (p. 75) and Amazon Polly API Reference (p. 56).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the Identity-based policy applies to. For more information, see Amazon Polly Resources and Operations (p. 75).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use `polly:PutLexicon` to add a lexicon to the region.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only). Amazon Polly doesn't support resource-based policies.

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

For a list showing all of the Amazon Polly API operations and the resources that they apply to, see Amazon Polly API Permissions: Actions, Permissions, and Resources Reference (p. 82).

Specifying Conditions in a Policy

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

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon Polly. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

Using Identity-Based Policies (IAM Policies) for Amazon Polly

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

> **Important**
> We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your Amazon Polly resources. For more information, see Overview of Managing Access Permissions to Your Amazon Polly Resources (p. 75).

Topics

- Permissions Required to Use the Amazon Polly Console (p. 79)
- AWS Managed (Predefined) Policies for Amazon Polly (p. 79)
- Customer Managed Policy Examples (p. 80)

The following shows an example of a permissions policy.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowGet-Delete-ListActions",
      "Effect": "Allow",
      "Action": [
         "polly:GetLexicon",
         "polly:DeleteLexicon",
         "polly:ListLexicons"],
      "Resource": "*"
      }
   ],
    "Statement": [{
      "Sid": "NoOverrideMyLexicons",
      "Effect": "Deny",
      "Action": [
         "polly:PutLexicon"],
      "Resource": "arn:aws:polly:us-east-1:123456789012:lexicon/my*"
      }
   ]
}
```

The policy has two statements:

- The first statement grants permission for three Polly actions (`polly:GetLexicon`, `polly:DeleteLexicon`, and `polly:ListLexicons` on any lexicon. Use of the wildcard character (*) as the resource grants universal permissions for these actions across all regions and lexicons owned by this account.
- The second statement explicitly denies permission for one Polly action (`polly:PutLexicon`). The ARN shown as the resource specifically applies this permission all lexicons that begin with the letters "my" that are in the region `us-east-1`.

For a table showing all of the Amazon Polly API operations and the resources that they apply to, see Amazon Polly API Permissions: Actions, Permissions, and Resources Reference (p. 82).

Permissions Required to Use the Amazon Polly Console

For a user to work with the Amazon Polly console, that user must have a minimum set of permissions that allows users to describe the Amazon Polly resources in their AWS account.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the Amazon Polly API.

To use the Amazon Polly console, you need to grant permissions to all the Amazon Polly APIs. There are no additional permissions needed. The following permissions policy is all that is needed to use the Amazon Polly console.

```
}
"Version": "2012-10-17",
   "Statement": [{
      "Sid": "Console-AllowAllPollyActions",
      "Effect": "Allow",
      "Action": [
         "polly:*"],
      "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).
- **AmazonPollyFullAccess** – Grants full access to resources and all the supported operations.

> **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.

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 are using AWS SDKs or the AWS CLI. When you are using the console, you need to grant permissions to all the Amazon Polly APIs. This is discussed in Permissions Required to Use the Amazon Polly Console (p. 79).

> **Note**
> All examples use the us-east-1 region and contain fictitious account IDs.

Examples

- Example 1: Allow All Amazon Polly Actions (p. 80)
- Example 2: Allow All Polly Actions Except DeleteLexicon (p. 80)
- Example 3: Allow DeleteLexicon (p. 81)
- Example 4: Allow Delete Lexicon in a Specified Region (p. 81)
- Example 5: Allow DeleteLexicon for Specified Lexicon (p. 82)

Example 1: Allow All Amazon Polly Actions

After you sign up (see Step 1.1: Sign up for AWS (p. 4)) you create an administrator user to manage your account, including creating users and managing their permissions.

You might choose to create a user who has permissions for all Amazon 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.

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

Example 2: Allow All 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.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Sid": "AllowAllActions-DenyDelete",
      "Effect": "Allow",
      "Action": [
         "polly:DescribeVoices",
         "polly:GetLexicon",
         "polly:PutLexicon",
         "polly:SynthesizeSpeech",
         "polly:ListLexicons"],
      "Resource": "*"
      }
      {
      "Sid": "DenyDeleteLexicon",
      "Effect": "Deny",
      "Action": [
         "polly:DeleteLexicon"],
      "Resource": "*"
      }
   ]
}
```

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.

```
{
  "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-1).

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Sid": "AllowDeleteSpecifiedRegion",
      "Effect": "Allow",
      "Action": [
         "polly:DeleteLexicon"],
      "Resource": "arn:aws:polly:us-east-1:123456789012: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-1).

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Sid": "AllowDeleteForSpecifiedLexicon",
      "Effect": "Allow",
      "Action": [
         "polly:DeleteLexicon"],
      "Resource": "arn:aws:polly:us-east-1:123456789012:lexicon/myLexicon"
      }
   ]
}
```

Amazon Polly API Permissions: Actions, Permissions, and Resources Reference

When you are setting up Access Control (p. 74) and writing 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-1`*`:`*`account-id`*`:lexicon/*` as the `Resource` value specifies permissions on all owned lexicons within the `us-east-1`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 (p. 57)
Required Permissions (API Action): `polly:DeleteLexicon`

Resources: `arn:aws:polly:`*`region`*`:`*`account-id`*`:lexicon/`*`LexiconName`*

API Operation: DescribeVoices (p. 58)
Required Permissions (API Action): `polly:DescribeVoices`

Resources: `arn:aws:polly:`*`region`*`:`*`account-id`*`:lexicon/`*`voice-name`*

API Operation: GetLexicon (p. 60)
Required Permissions (API Action): `polly:GetLexicon`

Resources: `arn:aws:polly:`*`region`*`:`*`account-id`*`:lexicon/`*`voice-name`*

API Operation: ListLexicons (p. 62)
Required Permissions (API Action): `polly:ListLexicons`

Resources: `arn:aws:polly:`*`region`*`:`*`account-id`*`:lexicon/*`

API Operation: PutLexicon (p. 64)
Required Permissions (API Action): `polly:ListLexicons`

Resources: *

API Operation: SynthesizeSpeech (p. 66)
Required Permissions (API Action): `polly:SynthesizeSpeech`

Resources: *

Document History for Amazon Polly

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

- **Latest documentation update:** November 30, 2016

Change	Description	Date
New service and guide	This is the initial release of the AWS Text-to-Speech service, Amazon Polly, and the *Amazon Polly Developer Guide*.	November 30, 2016

AWS Glossary

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