

Contents

[Speech service documentation](#)

[Overview](#)

[What is the Speech service?](#)

[Language and region support](#)

[Speech-to-text](#)

[Overview](#)

[What is speech-to-text?](#)

[Quickstarts](#)

[Recognize speech from an audio file](#)

[Recognize speech with microphone input](#)

[Recognize speech stored in blob storage](#)

[How-to guides](#)

[Change speech recognition source language](#)

[Improve accuracy with Custom Speech](#)

[Improve accuracy with Phrase Lists](#)

[Improve accuracy with Tenant Models](#)

[Select a specific microphone input device](#)

[Use automatic source language detection](#)

[Use compressed audio input formats](#)

[Use compressed audio input formats \(iOS\)](#)

[Use compressed audio input formats \(Android\)](#)

[Use speech recognition with Docker containers](#)

[Reference](#)

[Speech SDK APIs](#)

[C++](#)

[C#](#)

[Java](#)

[JavaScript](#)

[Objective-C](#)

[Python](#)

[Release notes](#)

[REST APIs](#)

[Speech-to-text REST API](#)

[Batch transcription REST API](#)

[Resources](#)

[Speech-to-text FAQ](#)

[Text-to-speech](#)

[Overview](#)

[What is text-to-speech?](#)

[Quickstarts](#)

[Synthesize speech into an audio file](#)

[Synthesize speech to a speaker](#)

[Async synthesis for long-form audio](#)

[How-to guides](#)

[Improve output with SSML](#)

[Improve output with Custom Voices](#)

[Synthesize speech with Long Audio API](#)

[Use speech synthesis with Docker containers](#)

[Reference](#)

[Speech SDK](#)

[C++](#)

[C#](#)

[Java](#)

[JavaScript](#)

[Objective-C](#)

[Python](#)

[Release notes](#)

[REST APIs](#)

[Text-to-speech REST API](#)

[Resources](#)

[Text-to-speech FAQ](#)

Intent recognition

Quickstarts

Recognize speech, intents, and entities

How-to guides

Recognize speech intents with LUIS, C#

Reference

Speech SDK

C++

C#

Java

JavaScript

Objective-C

Python

Release notes

LUIS Service

LUIS Developer Resources

Speech translation

Overview

What is speech translation?

Quickstarts

Translate speech-to-text

Translate speech to multiple target languages

Translate speech-to-speech

Tutorials

Build a Flask app to translate and synthesize text, REST

Reference

Speech SDK

C++

C#

Java

JavaScript

Objective-C

[Python](#)

[Release notes](#)

[Resources](#)

[Speech translation FAQ](#)

[Conversation transcription](#)

[Overview](#)

[What is conversation transcription?](#)

[How-to guides](#)

[Real time Conversation Transcription](#)

[Asynchronous Conversation Transcription](#)

[Reference](#)

[Speech SDK](#)

[C++](#)

[C#](#)

[Java](#)

[JavaScript](#)

[Objective-C](#)

[Python](#)

[Release notes](#)

[REST APIs](#)

[Conversation transcription REST APIs](#)

[Voice assistants](#)

[Overview](#)

[What is a voice assistant?](#)

[Building your assistant](#)

[Custom Commands \(Preview\)](#)

[Direct Line Speech](#)

[Quickstarts](#)

[C# \(UWP\)](#)

[Java \(Windows, macOS, Linux\)](#)

[Java \(Android\)](#)

[Create a Custom Command \(Preview\)](#)

[Create a Custom Command with parameters \(Preview\)](#)

[Use Custom Commands with Custom Voice \(Preview\)](#)

[Use Custom Commands with the Speech SDK \(Preview\)](#)

How-to guides

[Fulfill Custom Commands with the Speech SDK](#)

[Add parameter validation to Custom Commands](#)

[Add a confirmation to a Custom Command](#)

[Add a one-step correction to a Custom Command](#)

Tutorials

[Voice enable your bot with Speech SDK](#)

Resources

[Voice assistants FAQ](#)

Customization

Speech-to-text

[Custom Speech portal](#)

[Get started with Custom Speech](#)

[Prepare data](#)

[Inspect data](#)

[Evaluate accuracy](#)

[Train a custom model](#)

[Deploy a custom model](#)

[Guide Create human-labeled transcriptions](#)

Tenant Models

[Create, publish, and use a model](#)

Text-to-speech

[Responsible deployment](#)

[Gating of Custom Voice](#)

[Deployment of synthetic technology](#)

[Disclosure for voice talent](#)

[Disclosure design guidelines](#)

[Disclosure design patterns](#)

[Code of Conduct](#)

Custom Voice portal

[Get started with Custom Voice](#)

[Prepare data](#)

[Create and use Custom Voice models](#)

[Record voice samples](#)

Audio Content Creation

[How-to: Audio Content Creation](#)

[Custom keyword naming guidelines](#)

[Custom keyword spotting](#)

Speech Service

[Scenarios](#)

[Call centers](#)

[How-to guides](#)

[Try Speech Service for free](#)

[Create Speech Service Resource](#)

[Containers](#)

[Install and run containers](#)

[Configure containers](#)

[Kubernetes and Helm](#)

[Azure container instances](#)

[Samples](#)

[Text-to-speech REST samples](#)

[Batch transcription REST samples](#)

Reference

[REST APIs](#)

[Batch transcription & customization REST API](#)

[Conversation Transcription REST API](#)

[Speech-to-text REST API](#)

[Text-to-speech REST API](#)

[Swagger documentation](#)

Speech SDK

[Overview](#)

[About the Speech SDK](#)

[Scenario availability](#)

[Quickstarts](#)

[Create a project](#)

[Setup Speech SDK](#)

[How-to guides](#)

[How to enable logging](#)

[How to track memory usage](#)

[Samples](#)

[Speech SDK Samples \(GitHub\)](#)

[Speech Device SDK Samples \(GitHub\)](#)

[Reference](#)

[C++](#)

[C#](#)

[Java](#)

[JavaScript](#)

[Objective-C](#)

[Python](#)

[Speech SDK Release notes](#)

[Devices](#)

[Overview](#)

[About the Speech Devices SDK](#)

[Get the Speech Devices SDK](#)

[Hardware](#)

[Microphone array recommendations](#)

[Roobo v1 kit](#)

[Quickstarts](#)

[Windows](#)

[Linux](#)

[Android](#)

[Resources](#)

[Troubleshoot the Speech Devices SDK](#)

Reference

[Speech Devices SDK Release notes](#)

Migration

[From Bing Speech](#)

[From the Custom Speech service](#)

[From the Translator Speech API](#)

Resources

[Support](#)

[Regions](#)

[Pricing and limits](#)

[Compliance](#)

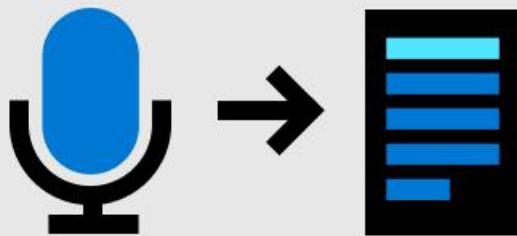
[Azure Roadmap](#)

[Privacy & cookies](#)

Speech Service documentation

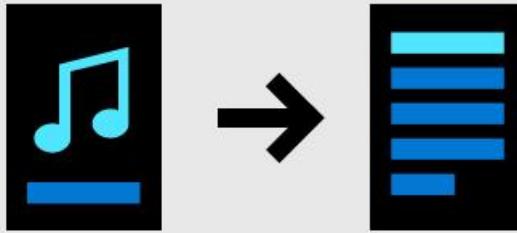
[Get Started](#)

[Speech-to-text](#)



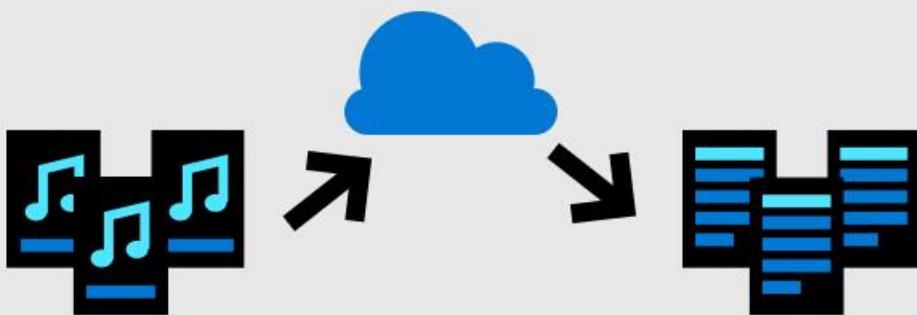
Recognize speech with a microphone

Use the Speech SDK to recognize speech from a microphone and transcribe the output.



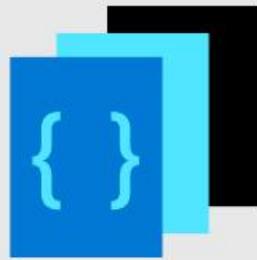
Recognize speech from an audio file

Use the Speech SDK to recognize speech from a single file and transcribe the output.



Asynchronous recognition from blob

Use our REST service to asynchronously recognize speech from files stored in Azure Blob Storage.



Language support

Learn more about programming and spoken language support for speech-to-text.



Pricing

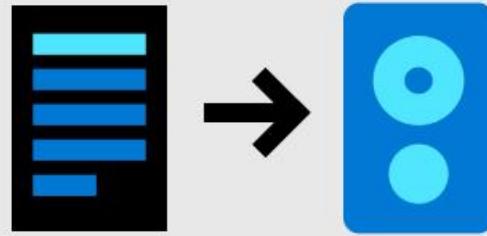
Learn more about the costs associated with speech-to-text.



Read the docs

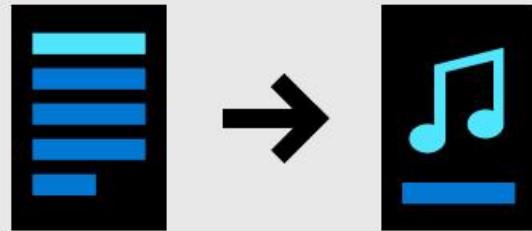
Learn how to add speech recognition to your apps, tools, and products. Includes concepts, tutorials, API reference, and release notes.

[Text-to-speech](#)



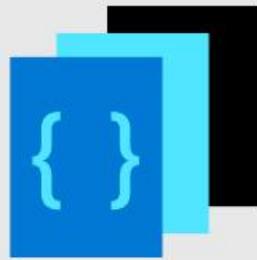
Synthesize speech to a speaker

Use the Speech SDK to synthesize speech to an audio output, like a speaker.



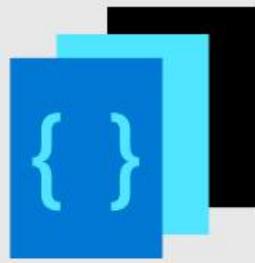
Synthesize speech to an audio file

Use the Speech SDK to synthesize speech to an audio file.



Speech Synthesis Markup Language

Use Speech Synthesis Markup Language to fine tune pitch, prosody, and speaking rate of your audio output.



Language support

Learn what languages are supported for speech synthesis.



Pricing

Learn more about the costs associated with text-to-speech.



Read the docs

Learn how to add speech synthesis to your apps, tools, and products. Includes concepts, tutorials, API reference, and release notes.

[Intent recognition](#)



Recognize speech, intents, and entities

Use the Speech SDK and Language Understanding (LUIS) to recognize speech, intents, and entities.



Language Understanding (LUIS) documentation

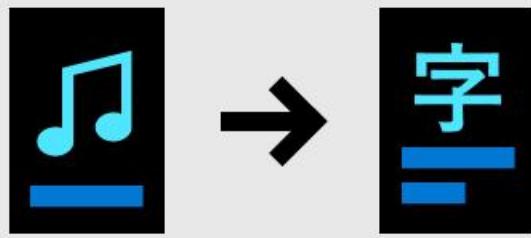
Learn more about the Language Understanding Service (LUIS) and Natural Language Processing (NLP).



Language Understanding (LUIS) portal

Build natural language into apps, bots, and IoT devices.

Speech translation



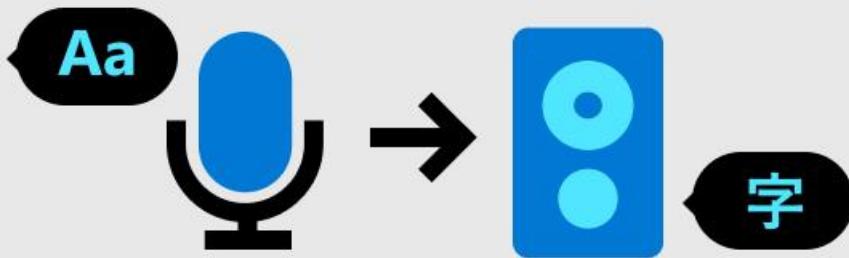
Translate speech-to-text from microphone

Use the Speech SDK to translate speech-to-text from a microphone.



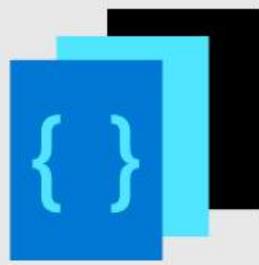
Translate speech to multiple target languages

Use the Speech SDK to translate speech to multiple target language outputs.



Translate speech-to-speech

Use the Speech SDK to translate speech to speech.



Language support

Learn more about programming and spoken language support for speech translation.



Pricing

Learn more about the costs associated with speech translation.



Read the docs

Learn how to add speech translation to your apps, tools, and products. Includes concepts, tutorials, API reference, and release notes.

[Conversation transcription](#)



Overview

Learn more about Conversation Transcription and how to integrate it into your products.



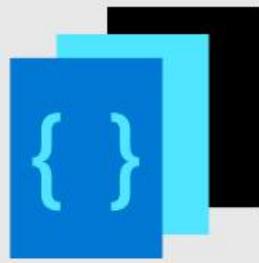
Transcribe a real-time conversation

Learn how to transcribe a conversation in real time.



Asynchronously transcribe conversations

Learn how to asynchronously transcribe conversations, poll for status, and download the outputs.



Language support

Learn more about programming and spoken language support for conversation transcription.



Pricing

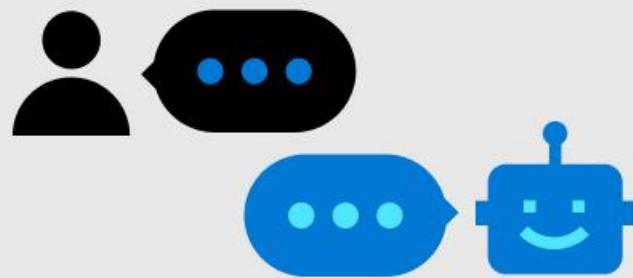
Learn more about the costs associated with Conversation Transcription.

Voice assistants



Overview

Learn more about what it takes to build a voice assistant.



[Integrate with Bot Framework](#)

Learn more about the Bot Framework, Direct Line Speech Channel, and more.



[Use Custom Commands](#)

Learn to easily build robust command and control voice applications, so that users can complete tasks by using their voice.

[Support](#)



[GitHub issues](#)

Browse open issues and/or create new issues for the Speech SDK on GitHub.



Stack Overflow

Ask questions, and get help from the Speech service community on Stack Overflow.



UserVoice forum

Share your ideas, suggest enhancements, or request new features for the Speech service.

Customization
Speech-to-text



Improve accuracy with Phrase Lists

Use Phrase Lists in the Speech SDK to improve recognition accuracy.



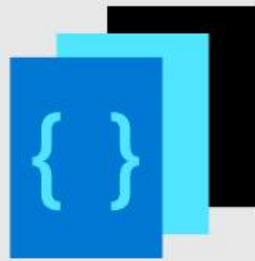
Improve accuracy with Tenant Models

Generate custom models with Office365 data to optimize speech recognition accuracy for organization-specific terms.



Improve accuracy with Custom Speech

A set of online tools that allow you to evaluate and improve Microsoft's speech-to-text accuracy for your apps, tools, and products.



Language support

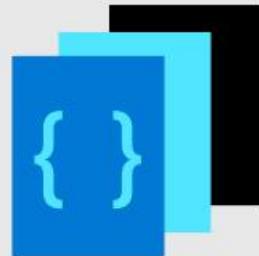
Learn more about programming and spoken language support for Custom Speech.



Pricing

Learn more about the costs associated with Custom Speech.

Text-to-speech



Improve synthesis with SSML

Use Speech Synthesis Markup Language to fine tune pitch, prosody, and speaking rate of your audio output.



Improve synthesis with Custom Voice

Build a recognizable, one-of-a-kind voice for your Text-to-Speech apps with your speaking data available.



Improve synthesis with Audio Content Creation

Use the Audio Content Creation tool to fine-tune your synthesized voice outputs.



Get access to Custom Neural Voices

This is a gated feature that allows you to create a Custom Neural Voice. Learn more about access and restrictions.



Pricing

Learn more about the costs associated with Custom Voice.

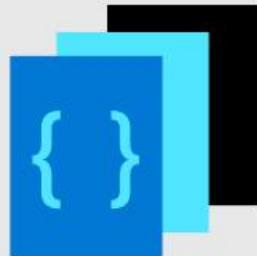
[Scenarios](#)

[Use cases](#)



Call center transcription

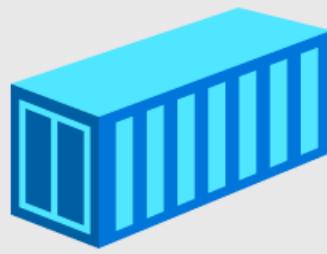
A common scenario for speech-to-text is transcribing large volumes of telephony data that may come from sources like an Interactive Voice Response (IVR) system.



Scenario & feature availability

Learn scenario and feature availability by platform and programming language.

Deployment



Deploy Speech service to containers

Use Docker to deploy the Speech service to a container instance.



Supported regions

Learn where the Speech service is supported.

What is the Speech service?

1/2/2020 • 4 minutes to read • [Edit Online](#)

The Speech service is the unification of speech-to-text, text-to-speech, and speech-translation into a single Azure subscription. It's easy to speech enable your applications, tools, and devices with the [Speech SDK](#), [Speech Devices SDK](#), or [REST APIs](#).

IMPORTANT

The Speech service has replaced Bing Speech API, Translator Speech, and Custom Speech. See *How-to guides > Migration* for migration instructions.

These features make up the Speech service. Use the links in this table to learn more about common use cases for each feature or browse the API reference.

Service	Feature	Description	SDK	REST
Speech-to-Text	Speech-to-text	Speech-to-text transcribes audio streams to text in real time that your applications, tools, or devices can consume or display. Use speech-to-text with Language Understanding (LUIS) to derive user intents from transcribed speech and act on voice commands.	Yes	Yes
	Batch Transcription	Batch transcription enables asynchronous speech-to-text transcription of large volumes of data. This is a REST-based service, which uses same endpoint as customization and model management.	No	Yes
	Conversation Transcription	Enables real-time speech recognition, speaker identification, and diarization. It's perfect for transcribing in-person meetings with the ability to distinguish speakers.	Yes	No

Service	Feature	Description	SDK	REST
	Create Custom Speech Models	If you are using speech-to-text for recognition and transcription in a unique environment, you can create and train custom acoustic, language, and pronunciation models to address ambient noise or industry-specific vocabulary.	No	Yes
Text-to-Speech	Text-to-speech	Text-to-speech converts input text into human-like synthesized speech using Speech Synthesis Markup Language (SSML) . Choose from standard voices and neural voices (see Language support).	Yes	Yes
	Create Custom Voices	Create custom voice fonts unique to your brand or product.	No	Yes
Speech Translation	Speech translation	Speech translation enables real-time, multi-language translation of speech to your applications, tools, and devices. Use this service for speech-to-speech and speech-to-text translation.	Yes	No

Service	Feature	Description	SDK	REST
Voice assistants	Voice assistants	Voice assistants using the Speech service empower developers to create natural, human-like conversational interfaces for their applications and experiences. The voice assistant service provides fast, reliable interaction between a device and an assistant implementation that uses the Bot Framework's Direct Line Speech channel or the integrated Custom Commands (Preview) service for task completion.	Yes	No

News and updates

Learn what's new with the Speech service.

- November 2019
 - Added two new speaking styles, `newscast` and `customerservice` for the `en-US-JessaNeural` voice.
- September 2019
 - Released Speech SDK 1.7.0. For a full list of updates, enhancements, and known issues, see [Release notes](#).
- August 2019
 - **New tutorial:** [Voice enable your bot with the Speech SDK, C#](#)
 - Added a new speaking style, `chat`, for the `en-US-JessaNeural` voice.
- June 2019
 - Released Speech SDK 1.6.0. For a full list of updates, enhancements, and known issues, see [Release notes](#).
- May 2019 - Documentation is now available for [Conversation Transcription](#), [Call Center Transcription](#), and [voice assistants](#).
- May 2019
 - Released Speech SDK 1.5.1. For a full list of updates, enhancements, and known issues, see [Release notes](#).
 - Released Speech SDK 1.5.0. For a full list of updates, enhancements, and known issues, see [Release notes](#).

Try the Speech service

We offer quickstarts in most popular programming languages, each designed to have you running code in less than 10 minutes. This table contains the most popular quickstarts for each feature. Use the left-hand navigation to explore additional languages and platforms.

SPEECH-TO-TEXT (SDK)	TEXT-TO-SPEECH (SDK)	TRANSLATION (SDK)
Recognize speech from an audio file	Synthesize speech into an audio file	Translate speech to text
Recognize speech with a microphone	Synthesize speech to a speaker	Translate speech to multiple target languages
Recognize speech stored in blob storage	Async synthesis for long-form audio	Translate speech-to-speech

NOTE

Speech-to-text and text-to-speech also have REST endpoints and associated quickstarts.

After you've had a chance to use the Speech service, try our tutorial that teaches you how to recognize intents from speech using the Speech SDK and LUIS.

- [Tutorial: Recognize intents from speech with the Speech SDK and LUIS, C#](#)
- [Tutorial: Voice enable your bot with the Speech SDK, C#](#)
- [Tutorial: Build a Flask app to translate text, analyze sentiment, and synthesize translated text to speech, REST](#)

Get sample code

Sample code is available on GitHub for the Speech service. These samples cover common scenarios like reading audio from a file or stream, continuous and single-shot recognition, and working with custom models. Use these links to view SDK and REST samples:

- [Speech-to-text, text-to-speech, and speech translation samples \(SDK\)](#)
- [Batch transcription samples \(REST\)](#)
- [Text-to-speech samples \(REST\)](#)
- [Voice assistant samples \(SDK\)](#)

Customize your speech experience

The Speech service works well with built-in models, however, you may want to further customize and tune the experience for your product or environment. Customization options range from acoustic model tuning to unique voice fonts for your brand.

SPEECH SERVICE	PLATFORM	DESCRIPTION
Speech-to-Text	Custom Speech	Customize speech recognition models to your needs and available data. Overcome speech recognition barriers such as speaking style, vocabulary and background noise.
Text-to-Speech	Custom Voice	Build a recognizable, one-of-a-kind voice for your Text-to-Speech apps with your speaking data available. You can further fine-tune the voice outputs by adjusting a set of voice parameters.

Reference docs

- [Speech SDK](#)
- [Speech Devices SDK](#)
- [REST API: Speech-to-text](#)
- [REST API: Text-to-speech](#)
- [REST API: Batch transcription and customization](#)

Next steps

[Get a Speech service subscription key for free](#)

Language and region support for the Speech service

12/23/2019 • 9 minutes to read • [Edit Online](#)

Language support varies by Speech service functionality. The following tables summarize language support for [Speech-to-text](#), [Text-to-speech](#), and [Speech translation](#) service offerings.

Speech-to-text

Both the Microsoft Speech SDK and the REST API support the following languages (locales). To improve accuracy, customization is offered for a subset of the languages through uploading Audio + Human-labeled Transcripts or Related Text: Sentences. Pronunciation customization is currently only available for `en-US` and `de-DE`. Learn more about customization [here](#).

Locale	Language	Supported	Customizable
<code>ar-EG</code>	Arabic (Egypt), modern standard	Yes	Yes
<code>ar-SA</code>	Arabic (Saudi Arabia)	Yes	Yes
<code>ar-AE</code>	Arabic (UAE)	Yes	Yes
<code>ar-KW</code>	Arabic (Kuwait)	Yes	Yes
<code>ar-QA</code>	Arabic (Qatar)	Yes	Yes
<code>ca-ES</code>	Catalan	Yes	No
<code>da-DK</code>	Danish (Denmark)	Yes	No
<code>de-DE</code>	German (Germany)	Yes	Yes
<code>en-AU</code>	English (Australia)	Yes	Yes
<code>en-CA</code>	English (Canada)	Yes	Yes
<code>en-GB</code>	English (United Kingdom)	Yes	Yes
<code>en-IN</code>	English (India)	Yes	Yes
<code>en-NZ</code>	English (New Zealand)	Yes	Yes
<code>en-US</code>	English (United States)	Yes	Yes
<code>es-ES</code>	Spanish (Spain)	Yes	Yes
<code>es-MX</code>	Spanish (Mexico)	Yes	Yes

Locale	Language	Supported	Customizable
fi-FI	Finnish (Finland)	Yes	No
fr-CA	French (Canada)	Yes	Yes
fr-FR	French (France)	Yes	Yes
gu-IN	Gujarati (Indian)	Yes	Yes
hi-IN	Hindi (India)	Yes	Yes
it-IT	Italian (Italy)	Yes	Yes
ja-JP	Japanese (Japan)	Yes	Yes
ko-KR	Korean (Korea)	Yes	Yes
mr-IN	Marathi (India)	Yes	Yes
nb-NO	Norwegian (Bokmål) (Norway)	Yes	No
nl-NL	Dutch (Netherlands)	Yes	Yes
pl-PL	Polish (Poland)	Yes	No
pt-BR	Portuguese (Brazil)	Yes	Yes
pt-PT	Portuguese (Portugal)	Yes	Yes
ru-RU	Russian (Russia)	Yes	Yes
sv-SE	Swedish (Sweden)	Yes	No
ta-IN	Tamil (India)	Yes	Yes
te-IN	Telugu (India)	Yes	Yes
zh-CN	Chinese (Mandarin, simplified)	Yes	Yes
zh-HK	Chinese (Cantonese, Traditional)	Yes	Yes
zh-TW	Chinese (Taiwanese Mandarin)	Yes	Yes
th-TH	Thai (Thailand)	Yes	No
tr-TR	Turkish	Yes	Yes

Text-to-speech

Both the Microsoft Speech SDK and REST APIs support these voices, each of which supports a specific language and dialect, identified by locale.

IMPORTANT

Pricing varies for standard, custom and neural voices. Please visit the [Pricing](#) page for additional information.

Neural voices

Neural text-to-speech is a new type of speech synthesis powered by deep neural networks. When using a neural voice, synthesized speech is nearly indistinguishable from the human recordings.

Neural voices can be used to make interactions with chatbots and voice assistants more natural and engaging, convert digital texts such as e-books into audiobooks and enhance in-car navigation systems. With the human-like natural prosody and clear articulation of words, neural voices significantly reduce listening fatigue when users interact with AI systems.

For more information about regional availability, see [regions](#).

LOCALE	LANGUAGE	GENDER	FULL SERVICE NAME MAPPING	SHORT VOICE NAME
de-DE	German (Germany)	Female	"Microsoft Server Speech Text to Speech Voice (de-DE, KatjaNeural)"	"de-DE-KatjaNeural"
en-US	English (US)	Male	"Microsoft Server Speech Text to Speech Voice (en-US, GuyNeural)"	"en-US-GuyNeural"
en-US	English (US)	Female	"Microsoft Server Speech Text to Speech Voice (en-US, JessaNeural)"	"en-US-JessaNeural"
it-IT	Italian (Italy)	Female	"Microsoft Server Speech Text to Speech Voice (it-IT, ElsaNeural)"	"it-IT-ElsaNeural"
zh-CN	Chinese (Mainland)	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, XiaoxiaoNeural)"	"zh-CN-XiaoxiaoNeural"

To learn how you can configure and adjust neural voices, see [Speech synthesis markup language](#).

NOTE

You can use either the full service name mapping or the short voice name in your speech synthesis requests.

Standard voices

More than 75 standard voices are available in over 45 languages and locales, which allow you to convert text into

synthesized speech. For more information about regional availability, see [regions](#).

Locale	Language	Gender	Full Service Name Mapping	Short Name
+ ar-EG	Arabic (Egypt)	Female	"Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)"	"ar-EG-Hoda"
ar-SA	Arabic (Saudi Arabia)	Male	"Microsoft Server Speech Text to Speech Voice (ar-SA, Naayf)"	"ar-SA-Naayf"
bg-BG	Bulgarian	Male	"Microsoft Server Speech Text to Speech Voice (bg-BG, Ivan)"	"bg-BG-Ivan"
ca-ES	Catalan (Spain)	Female	"Microsoft Server Speech Text to Speech Voice (ca-ES, HerenaRUS)"	"ca-ES-HerenaRUS"
cs-CZ	Czech	Male	"Microsoft Server Speech Text to Speech Voice (cs-CZ, Jakub)"	"cs-CZ-Jakub"
da-DK	Danish	Female	"Microsoft Server Speech Text to Speech Voice (da-DK, HelleRUS)"	"da-DK-HelleRUS"
de-AT	German (Austria)	Male	"Microsoft Server Speech Text to Speech Voice (de-AT, Michael)"	"de-AT-Michael"
de-CH	German (Switzerland)	Male	"Microsoft Server Speech Text to Speech Voice (de-CH, Karsten)"	"de-CH-Karsten"
de-DE	German (Germany)	Female	"Microsoft Server Speech Text to Speech Voice (de-DE, Hedda)"	"de-DE-Hedda"
		Female	"Microsoft Server Speech Text to Speech Voice (de-DE, HeddaRUS)"	"de-DE-HeddaRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (de-DE, Stefan, Apollo)"	"de-DE-Stefan-Apollo"
el-GR	Greek	Male	"Microsoft Server Speech Text to Speech Voice (el-GR, Stefanos)"	"el-GR-Stefanos"

Locale	Language	Gender	Full Service Name Mapping	Short Name
en-AU	English (Australia)	Female	"Microsoft Server Speech Text to Speech Voice (en-AU, Catherine)"	"en-AU-Catherine"
		Female	"Microsoft Server Speech Text to Speech Voice (en-AU, HayleyRUS)"	"en-AU-HayleyRUS"
en-CA	English (Canada)	Female	"Microsoft Server Speech Text to Speech Voice (en-CA, Linda)"	"en-CA-Linda"
		Female	"Microsoft Server Speech Text to Speech Voice (en-CA, HeatherRUS)"	"en-CA-HeatherRUS"
en-GB	English (UK)	Female	"Microsoft Server Speech Text to Speech Voice (en-GB, Susan, Apollo)"	"en-GB-Susan-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (en-GB, HazelRUS)"	"en-GB-HazelRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (en-GB, George, Apollo)"	"en-GB-George-Apollo"
en-IE	English (Ireland)	Male	"Microsoft Server Speech Text to Speech Voice (en-IE, Sean)"	"en-IE-Sean"
en-IN	English (India)	Female	"Microsoft Server Speech Text to Speech Voice (en-IN, Heera, Apollo)"	"en-IN-Heera-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (en-IN, PriyaRUS)"	"en-IN-PriyaRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (en-IN, Ravi, Apollo)"	"en-IN-Ravi-Apollo"

Locale	Language	Gender	Full Service Name Mapping	Short Name
en-US	English (US)	Female	"Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)"	"en-US-ZiraRUS"
		Female	"Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)"	"en-US-JessaRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (en-US, BenjaminRUS)"	"en-US-BenjaminRUS"
		Female	"Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)"	"en-US-Jessa24kRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (en-US, Guy24kRUS)"	"en-US-Guy24kRUS"
es-ES	Spanish (Spain)	Female	"Microsoft Server Speech Text to Speech Voice (es-ES, Laura, Apollo)"	"es-ES-Laura-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (es-ES, HelenaRUS)"	"es-ES-HelenaRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (es-ES, Pablo, Apollo)"	"es-ES-Pablo-Apollo"
es-MX	Spanish (Mexico)	Female	"Microsoft Server Speech Text to Speech Voice (es-MX, HildaRUS)"	"es-MX-HildaRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (es-MX, Raul, Apollo)"	"es-MX-Raul-Apollo"
fi-FI	Finnish	Female	"Microsoft Server Speech Text to Speech Voice (fi-FI, HeidiRUS)"	"fi-FI-HeidiRUS"

LOCALE	LANGUAGE	GENDER	FULL SERVICE NAME MAPPING	SHORT NAME
fr-CA	French (Canada)	Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, Caroline)"	"fr-CA-Caroline"
		Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, HarmonieRUS)"	"fr-CA-HarmonieRUS"
fr-CH	French (Switzerland)	Male	"Microsoft Server Speech Text to Speech Voice (fr-CH, Guillaume)"	"fr-CH-Guillaume"
fr-FR	French (France)	Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, Julie, Apollo)"	"fr-FR-Julie-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, HortenseRUS)"	"fr-FR-HortenseRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (fr-FR, Paul, Apollo)"	"fr-FR-Paul-Apollo"
he-IL	Hebrew (Israel)	Male	"Microsoft Server Speech Text to Speech Voice (he-IL, Asaf)"	"he-IL-Asaf"
hi-IN	Hindi (India)	Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana, Apollo)"	"hi-IN-Kalpana-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana)"	"hi-IN-Kalpana"
		Male	"Microsoft Server Speech Text to Speech Voice (hi-IN, Hemant)"	"hi-IN-Hemant"
hr-HR	Croatian	Male	"Microsoft Server Speech Text to Speech Voice (hr-HR, Matej)"	"hr-HR-Matej"

Locale	Language	Gender	Full Service Name Mapping	Short Name
hu-HU	Hungarian	Male	"Microsoft Server Speech Text to Speech Voice (hu-HU, Szabolcs)"	"hu-HU-Szabolcs"
id-ID	Indonesian	Male	"Microsoft Server Speech Text to Speech Voice (id-ID, Andika)"	"id-ID-Andika"
it-IT	Italian	Male	"Microsoft Server Speech Text to Speech Voice (it-IT, Cosimo, Apollo)"	"it-IT-Cosimo-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (it-IT, LuciaRUS)"	"it-IT-LuciaRUS"
ja-JP	Japanese	Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ayumi, Apollo)"	"ja-JP-Ayumi-Apollo"
		Male	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ichiro, Apollo)"	"ja-JP-Ichiro-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, HarukaRUS)"	"ja-JP-HarukaRUS"
ko-KR	Korean	Female	"Microsoft Server Speech Text to Speech Voice (ko-KR, HeamiRUS)"	"ko-KR-HeamiRUS"
ms-MY	Malay	Male	"Microsoft Server Speech Text to Speech Voice (ms-MY, Rizwan)"	"ms-MY-Rizwan"
nb-NO	Norwegian	Female	"Microsoft Server Speech Text to Speech Voice (nb-NO, HuldaRUS)"	"nb-NO-HuldaRUS"
nl-NL	Dutch	Female	"Microsoft Server Speech Text to Speech Voice (nl-NL, HannaRUS)"	"nl-NL-HannaRUS"

Locale	Language	Gender	Full Service Name Mapping	Short Name
pl-PL	Polish	Female	"Microsoft Server Speech Text to Speech Voice (pl-PL, PaulinaRUS)"	"pl-PL-PaulinaRUS"
pt-BR	Portuguese (Brazil)	Female	"Microsoft Server Speech Text to Speech Voice (pt-BR, HeloisaRUS)"	"pt-BR-HeloisaRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (pt-BR, Daniel, Apollo)"	"pt-BR-Daniel-Apollo"
pt-PT	Portuguese (Portugal)	Female	"Microsoft Server Speech Text to Speech Voice (pt-PT, HeliaRUS)"	"pt-PT-HeliaRUS"
ro-RO	Romanian	Male	"Microsoft Server Speech Text to Speech Voice (ro-RO, Andrei)"	"ro-RO-Andrei"
ru-RU	Russian	Female	"Microsoft Server Speech Text to Speech Voice (ru-RU, Irina, Apollo)"	"ru-RU-Irina-Apollo"
		Male	"Microsoft Server Speech Text to Speech Voice (ru-RU, Pavel, Apollo)"	"ru-RU-Pavel-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (ru-RU, EkaterinaRUS)"	ru-RU-EkaterinaRUS
sk-SK	Slovak	Male	"Microsoft Server Speech Text to Speech Voice (sk-SK, Filip)"	"sk-SK-Filip"
sl-SI	Slovenian	Male	"Microsoft Server Speech Text to Speech Voice (sl-SI, Lado)"	"sl-SI-Lado"
sv-SE	Swedish	Female	"Microsoft Server Speech Text to Speech Voice (sv-SE, HedvigRUS)"	"sv-SE-HedvigRUS"
ta-IN	Tamil (India)	Male	"Microsoft Server Speech Text to Speech Voice (ta-IN, Valluvar)"	"ta-IN-Valluvar"

LOCALE	LANGUAGE	GENDER	FULL SERVICE NAME MAPPING	SHORT NAME
te-IN	Telugu (India)	Female	"Microsoft Server Speech Text to Speech Voice (te-IN, Chitra)"	"te-IN-Chitra"
th-TH	Thai	Male	"Microsoft Server Speech Text to Speech Voice (th-TH, Pattara)"	"th-TH-Pattara"
tr-TR	Turkish	Female	"Microsoft Server Speech Text to Speech Voice (tr-TR, SedaRUS)"	"tr-TR-SedaRUS"
vi-VN	Vietnamese	Male	"Microsoft Server Speech Text to Speech Voice (vi-VN, An)"	"vi-VN-An"
zh-CN	Chinese (Mainland)	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, HuihuiRUS)"	"zh-CN-HuihuiRUS"
		Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, Yaoyao, Apollo)"	"zh-CN-Yaoyao-Apollo"
		Male	"Microsoft Server Speech Text to Speech Voice (zh-CN, Kangkang, Apollo)"	"zh-CN-Kangkang-Apollo"
zh-HK	Chinese (Hong Kong)	Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, Tracy, Apollo)"	"zh-HK-Tracy-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, TracyRUS)"	"zh-HK-TracyRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (zh-HK, Danny, Apollo)"	"zh-HK-Danny-Apollo"

Locale	Language	Gender	Full Service Name Mapping	Short Name
<code>zh-TW</code>	Chinese (Taiwan)	Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, Yating, Apollo)"	"zh-TW-Yating-Apollo"
		Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, HanHanRUS)"	"zh-TW-HanHanRUS"
		Male	"Microsoft Server Speech Text to Speech Voice (zh-TW, Zhiwei, Apollo)"	"zh-TW-Zhiwei-Apollo"

[†] *ar-EG* supports Modern Standard Arabic (MSA).

NOTE

You can use either the full service name mapping or the short voice name in your speech synthesis requests.

Customization

Voice customization is available for `de-DE`, `en-GB`, `en-IN`, `en-US`, `es-MX`, `fr-FR`, `it-IT`, `pt-BR`, and `zh-CN`. Select the right locale that matches the training data you have to train a custom voice model. For example, if the recording data you have is spoken in English with a British accent, select `en-GB`.

NOTE

We do not support bi-lingual model training in Custom Voice, except for the Chinese-English bi-lingual. Select "Chinese-English bilingual" if you want to train a Chinese voice that can speak English as well. Voice training in all locales starts with a data set of 2,000+ utterances, except for the `en-US` and `zh-CN` where you can start with any size of training data.

Speech translation

The **Speech Translation** API supports different languages for speech-to-speech and speech-to-text translation. The source language must always be from the Speech-to-text language table. The available target languages depend on whether the translation target is speech or text. You may translate incoming speech into more than [60 languages](#). A subset of languages are available for [speech synthesis](#).

Text languages

Text Language	Language Code
Afrikaans	<code>af</code>
Arabic	<code>ar</code>
Bangla	<code>bn</code>
Bosnian (Latin)	<code>bs</code>

TEXT LANGUAGE	LANGUAGE CODE
Bulgarian	bg
Cantonese (Traditional)	yue
Catalan	ca
Chinese Simplified	zh-Hans
Chinese Traditional	zh-Hant
Croatian	hr
Czech	cs
Danish	da
Dutch	nl
English	en
Estonian	et
Fijian	fj
Filipino	fil
Finnish	fi
French	fr
German	de
Greek	el
Haitian Creole	ht
Hebrew	he
Hindi	hi
Hmong Daw	mww
Hungarian	hu
Indonesian	id
Italian	it

TEXT LANGUAGE	LANGUAGE CODE
Japanese	ja
Kiswahili	sw
Klingon	tlh
Klingon (plqaD)	tlh-Qaak
Korean	ko
Latvian	lv
Lithuanian	lt
Malagasy	mg
Malay	ms
Maltese	mt
Norwegian	nb
Persian	fa
Polish	pl
Portuguese	pt
Queretaro Otomi	otq
Romanian	ro
Russian	ru
Samoan	sm
Serbian (Cyrillic)	sr-Cyr1
Serbian (Latin)	sr-Latn
Slovak	sk
Slovenian	sl
Spanish	es
Swedish	sv

TEXT LANGUAGE	LANGUAGE CODE
Tahitian	ty
Tamil	ta
Telugu	te
Thai	th
Tongan	to
Turkish	tr
Ukrainian	uk
Urdu	ur
Vietnamese	vi
Welsh	cy
Yucatec Maya	yua

Next steps

- [Get your Speech service trial subscription](#)
- [See how to recognize speech in C#](#)

What is speech-to-text?

12/16/2019 • 2 minutes to read • [Edit Online](#)

Speech-to-text from the Speech service, also known as speech recognition, enables real-time transcription of audio streams into text. Your applications, tools, or devices can consume, display, and take action on this text as command input. This service is powered by the same recognition technology that Microsoft uses for Cortana and Office products. It seamlessly works with the [translation](#) and [text-to-speech](#) service offerings. For a full list of available speech-to-text languages, see [supported languages](#).

The speech-to-text service defaults to using the Universal language model. This model was trained using Microsoft-owned data and is deployed in the cloud. It's optimal for conversational and dictation scenarios. When using speech-to-text for recognition and transcription in a unique environment, you can create and train custom acoustic, language, and pronunciation models. Customization is helpful for addressing ambient noise or industry-specific vocabulary.

NOTE

Bing Speech was decommissioned on October 15, 2019. If your applications, tools, or products are using the Bing Speech APIs or Custom Speech, we've created guides to help you migrate to the Speech service.

- [Migrate from Bing Speech to the Speech service](#)
- [Migrate from Custom Speech to the Speech service](#)

Get started with speech-to-text

The speech-to-text service is available via the [Speech SDK](#). There are several common scenarios available as quickstarts, in various languages and platforms:

- [Quickstart: Recognize speech with microphone input](#)
- [Quickstart: Recognize speech from a file](#)
- [Quickstart: Recognize speech stored in blob storage](#)

If you prefer to use the speech-to-text REST service, see [REST APIs](#).

Tutorials and sample code

After you've had a chance to use the Speech service, try our tutorial that teaches you how to recognize intents from speech using the Speech SDK and LUIS.

- [Tutorial: Recognize intents from speech with the Speech SDK and LUIS, using C#](#)

Sample code for the Speech SDK is available on GitHub. These samples cover common scenarios like reading audio from a file or stream, continuous and single-shot recognition, and working with custom models.

- [Speech-to-text samples \(SDK\)](#)
- [Batch transcription samples \(REST\)](#)

Customization

In addition to the standard Speech service model, you can create custom models. Customization helps to overcome speech recognition barriers such as speaking style, vocabulary and background noise, see [Custom](#)

[Speech](#). Customization options vary by language/locale, see [supported languages](#) to verify support.

Reference docs

The Speech service provides two SDKs. The first SDK is the primary [Speech SDK](#) and provides most of the functionalities needed to interact with the Speech service. The second SDK is specific to devices, appropriately named the [Speech Devices SDK](#). Both SDKs are available in many languages.

Speech SDK reference docs

Use the following list to find the appropriate Speech SDK reference docs:

- [C# SDK](#)
- [C++ SDK](#)
- [Java SDK](#)
- [Python SDK](#)
- [JavaScript SDK](#)
- [Objective-C SDK](#)

TIP

The Speech service SDK is actively maintained and updated. To track changes, updates and feature additions refer to the [Speech SDK release notes](#).

Speech Devices SDK reference docs

The [Speech Devices SDK](#) is a superset of the Speech SDK, with extended functionality for specific devices. To download the Speech Devices SDK, you must first [choose a development kit](#).

REST API references

For references of various Speech service REST APIs, refer to the listing below:

- [REST API: Speech-to-text](#)
- [REST API: Text-to-speech](#)
- [REST API: Batch transcription and customization](#)

Next steps

- [Get a Speech service subscription key for free](#)
- [Get the Speech SDK](#)

Quickstart: Recognize speech from an audio file

12/4/2019 • 19 minutes to read • [Edit Online](#)

In this quickstart you will use the [Speech SDK](#) to recognize speech from an audio file. After satisfying a few prerequisites, recognizing speech from a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `AudioConfig` object that specifies the .WAV file name.
- Create a `SpeechRecognizer` object using the `SpeechConfig` and `AudioConfig` objects from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Supported audio input format

The Speech SDK uses the following format for audio input.

FORMAT	CODEC	BITRATE	SAMPLE RATE	CHANNELS
WAV	PCM	16-bit	8 kHz or 16 kHz	1 (mono)

Open your project in Visual Studio

The first step is to make sure that you have your project open in Visual Studio.

1. Launch Visual Studio 2019.
2. Load your project and open `Program.cs`.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project. Make note that you've created an async method called `RecognizeSpeechAsync()`.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeSpeechAsync()
        {
        }

        static void Main()
        {
            RecognizeSpeechAsync().Wait();
        }
    }
}
```

Create a Speech configuration

Before you can initialize a `SpeechRecognizer` object, you need to create a configuration that uses your subscription key and subscription region. Insert this code in the `RecognizeSpeechAsync()` method.

NOTE

This sample uses the `FromSubscription()` method to build the `SpeechConfig`. For a full list of available methods, see [SpeechConfig Class](#). The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

```
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

Create an Audio configuration

Now, you need to create an `AudioConfig` object that points to your audio file. This object is created inside of a `using` statement to ensure the proper release of unmanaged resources. Insert this code in the `RecognizeSpeechAsync()` method, right below your Speech configuration.

```
using (var audioInput = AudioConfig.FromWavFileInput(@"whatstheweatherlike.wav"))
{}
```

Initialize a SpeechRecognizer

Now, let's create the `SpeechRecognizer` object using the `SpeechConfig` and `AudioConfig` objects created earlier. This object is also created inside of a `using` statement to ensure the proper release of unmanaged resources. Insert this code in the `RecognizeSpeechAsync()` method, inside the `using` statement that wraps your `AudioConfig` object.

```
using (var recognizer = new SpeechRecognizer(config, audioInput))
{
}
```

Recognize a phrase

From the `SpeechRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech.

Inside the using statement, add this code:

```
Console.WriteLine("Recognizing first result...");
var result = await recognizer.RecognizeOnceAsync();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, you'll want to do something with it. We're going to keep it simple and print the result to console.

Inside the using statement, below `RecognizeOnceAsync()`, add this code:

```
if (result.Reason == ResultReason.RecognizedSpeech)
{
    Console.WriteLine($"We recognized: {result.Text}");
}
else if (result.Reason == ResultReason.NoMatch)
{
    Console.WriteLine($"NOMATCH: Speech could not be recognized.");
}
else if (result.Reason == ResultReason.Canceled)
{
    var cancellation = CancellationDetails.FromResult(result);
    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

    if (cancellation.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }
}
```

Check your code

At this point, your code should look like this:

```

// Copyright (c) Microsoft. All rights reserved.
// Licensed under the MIT license. See LICENSE.md file in the project root for full license information.
//

using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeSpeechAsync()
        {
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            using (var audioInput = AudioConfig.FromWavFileInput(@"whatstheweatherlike.wav"))
            {
                using (var recognizer = new SpeechRecognizer(config, audioInput))
                {
                    Console.WriteLine("Recognizing first result...");
                    var result = await recognizer.RecognizeOnceAsync();

                    if (result.Reason == ResultReason.RecognizedSpeech)
                    {
                        Console.WriteLine($"We recognized: {result.Text}");
                    }
                    else if (result.Reason == ResultReason.NoMatch)
                    {
                        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
                    }
                    else if (result.Reason == ResultReason.Canceled)
                    {
                        var cancellation = CancellationDetails.FromResult(result);
                        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

                        if (cancellation.Reason == CancellationReason.Error)
                        {
                            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                            Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
                            Console.WriteLine($"CANCELED: Did you update the subscription info?");
                        }
                    }
                }
            }

            static void Main()
            {
                RecognizeSpeechAsync().Wait();
            }
        }
    }
}

```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

- 1. Compile the code** - From the menu bar of Visual Studio, choose **Build > Build Solution**.
- 2. Start your app** - From the menu bar, choose **Debug > Start Debugging** or press **F5**.
- 3. Start recognition** - Your audio file is sent to the Speech service, transcribed as text, and rendered in the console.

```
Recognizing first result...
We recognized: What's the weather like?
```

Next steps

[Explore C# samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to recognize speech from an audio file. After satisfying a few prerequisites, recognizing speech from a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `AudioConfig` object that specifies the .WAV file name.
- Create a `SpeechRecognizer` object using the `SpeechConfig` and `AudioConfig` objects from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [Linux](#)
- [macOS](#)
- [Windows](#)

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Supported audio input format

The Speech SDK uses the following format for audio input.

FORMAT	CODEC	BITRATE	SAMPLE RATE	CHANNELS
WAV	PCM	16-bit	8 kHz or 16 kHz	1 (mono)

Add sample code

1. Create a C++ source file named `helloworld.cpp`, and paste the following code into it.

```

// Creates an instance of a speech config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
auto config = SpeechConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");

// Creates a speech recognizer using a WAV file. The default language is "en-us".
// Replace with your own audio file name.
auto audioInput = AudioConfig::FromWavFileInput("whatstheweatherlike.wav");
auto recognizer = SpeechRecognizer::FromConfig(config, audioInput);
cout << "Recognizing first result...\n";

// Starts speech recognition, and returns after a single utterance is recognized. The end of a
// single utterance is determined by listening for silence at the end or until a maximum of 15
// seconds of audio is processed. The task returns the recognition text as result.
// Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
// shot recognition like command or query.
// For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
auto result = recognizer->RecognizeOnceAsync().get();

// Checks result.
if (result->Reason == ResultReason::RecognizedSpeech)
{
    cout << "RECOGNIZED: Text=" << result->Text << std::endl;
}
else if (result->Reason == ResultReason::NoMatch)
{
    cout << "NOMATCH: Speech could not be recognized." << std::endl;
}
else if (result->Reason == ResultReason::Canceled)
{
    auto cancellation = CancellationDetails::FromResult(result);
    cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

    if (cancellation->Reason == CancellationReason::Error)
    {
        cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
        cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
        cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}

```

2. In this new file, replace the string `YourSubscriptionKey` with your Speech service subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
4. Replace the string `whatstheweatherlike.wav` with your own filename.

NOTE

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Build the app

NOTE

Make sure to enter the commands below as a *single command line*. The easiest way to do that is to copy the command by using the **Copy** button next to each command, and then paste it at your shell prompt.

- On an **x64** (64-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I  
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L  
"$SPEECHSDK_ROOT/lib/x64" -l:libasound.so.2
```

- On an **x86** (32-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I  
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L  
"$SPEECHSDK_ROOT/lib/x86" -l:libasound.so.2
```

- On an **ARM64** (64-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I  
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L  
"$SPEECHSDK_ROOT/lib/arm64" -l:libasound.so.2
```

Run the app

1. Configure the loader's library path to point to the Speech SDK library.

- On an **x64** (64-bit) system, enter the following command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/x64"
```

- On an **x86** (32-bit) system, enter this command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/x86"
```

- On an **ARM64** (64-bit) system, enter the following command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/arm64"
```

2. Run the application.

```
./helloworld
```

3. Your audio file is transmitted to the Speech service and the first utterance in the file is transcribed to text, which appears in the same window.

```
Recognizing first result...  
We recognized: What's the weather like?
```

Next steps

[Explore C++ samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to recognize speech from an audio file. After satisfying a few prerequisites, recognizing speech from a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `AudioConfig` object that specifies the .WAV file name.
- Create a `SpeechRecognizer` object using the `SpeechConfig` and `AudioConfig` objects from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

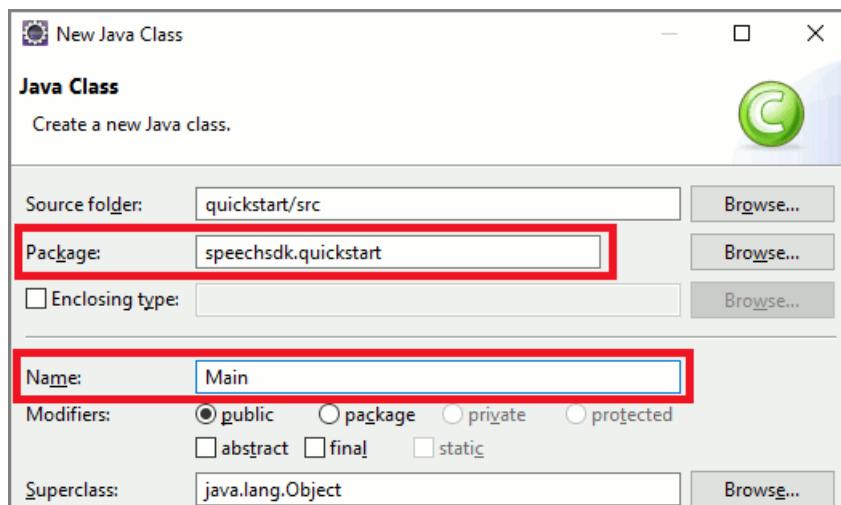
Supported audio input format

The Speech SDK uses the following format for audio input.

FORMAT	CODEC	BITRATE	SAMPLE RATE	CHANNELS
WAV	PCM	16-bit	8 kHz or 16 kHz	1 (mono)

Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in `Main.java` with the following snippet:

```
package speechsdk.quickstart;

import java.util.concurrent.Future;
import com.microsoft.cognitiveservices.speech.*;

/**
 * Quickstart: recognize speech using the Speech SDK for Java.
 */
```

```

public class Main {

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {
        try {
            // Replace below with your own subscription key
            String speechSubscriptionKey = "YourSubscriptionKey";
            // Replace below with your own service region (e.g., "westus").
            String serviceRegion = "YourServiceRegion";
            // Replace below with your own filename.
            String audioFileName = "whatstheweatherlike.wav";

            int exitCode = 1;
            SpeechConfig config = SpeechConfig.fromSubscription(speechSubscriptionKey, serviceRegion);
            assert(config != null);

            AudioConfig audioInput = AudioConfig.fromWavFileInput(audioFileName);
            assert(audioInput != null);

            SpeechRecognizer reco = new SpeechRecognizer(config, audioInput);
            assert(reco != null);

            System.out.println("Recognizing first result...");

            Future<SpeechRecognitionResult> task = reco.recognizeOnceAsync();
            assert(task != null);

            SpeechRecognitionResult result = task.get();
            assert(result != null);

            if (result.getReason() == ResultReason.RecognizedSpeech) {
                System.out.println("We recognized: " + result.getText());
                exitCode = 0;
            }
            else if (result.getReason() == ResultReason.NoMatch) {
                System.out.println("NOMATCH: Speech could not be recognized.");
            }
            else if (result.getReason() == ResultReason.Canceled) {
                CancellationDetails cancellation = CancellationDetails.fromResult(result);
                System.out.println("CANCELED: Reason=" + cancellation.getReason());

                if (cancellation.getReason() == CancellationReason.Error) {
                    System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
                    System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
                    System.out.println("CANCELED: Did you update the subscription info?");
                }
            }
            reco.close();

            System.exit(exitCode);
        } catch (Exception ex) {
            System.out.println("Unexpected exception: " + ex.getMessage());

            assert(false);
            System.exit(1);
        }
    }
}

```

4. Replace the string `YourSubscriptionKey` with your subscription key.
5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).

6. Replace the string `whatsttheweatherlike.wav` with your own filename.

7. Save changes to the project.

NOTE

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Build and run the app

Press F11, or select **Run > Debug**. The first 15 seconds of speech input from your audio file will be recognized and logged in the console window.

```
Recognizing first result...
We recognized: What's the weather like?
```

Next steps

[Explore Java samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to recognize speech from an audio file. After satisfying a few prerequisites, recognizing speech from a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `AudioConfig` object that specifies the .WAV file name.
- Create a `SpeechRecognizer` object using the `SpeechConfig` and `AudioConfig` objects from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Create a LUIS application and get an endpoint key](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Supported audio input format

The Speech SDK uses the following format for audio input.

FORMAT	CODEC	BITRATE	SAMPLE RATE	CHANNELS
WAV	PCM	16-bit	8 kHz or 16 kHz	1 (mono)

Support and updates

Updates to the Speech SDK Python package are distributed via PyPI and announced in the [Release notes](#). If a new version is available, you can update to it with the command

```
pip install --upgrade azure-cognitiveservices-speech
```

Check which version is currently installed by inspecting the `azure.cognitiveservices.speech.__version__` variable.

If you have a problem, or you're missing a feature, see [Support and help options](#).

Create a Python application that uses the Speech SDK

Run the sample

You can copy the [sample code](#) from this quickstart to a source file `quickstart.py` and run it in your IDE or in the console:

```
python quickstart.py
```

Or you can download this quickstart tutorial as a [Jupyter](#) notebook from the [Speech SDK sample repository](#) and run it as a notebook.

Sample code

NOTE

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

```

import azure.cognitiveservices.speech as speechsdk

# Creates an instance of a speech config with specified subscription key and service region.
# Replace with your own subscription key and service region (e.g., "westus").
speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"
speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)

# Creates an audio configuration that points to an audio file.
# Replace with your own audio filename.
audio_filename = "whatstheweatherlike.wav"
audio_input = speechsdk.AudioConfig(filename=audio_filename)

# Creates a recognizer with the given settings
speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config, audio_config=audio_input)

print("Recognizing first result...")

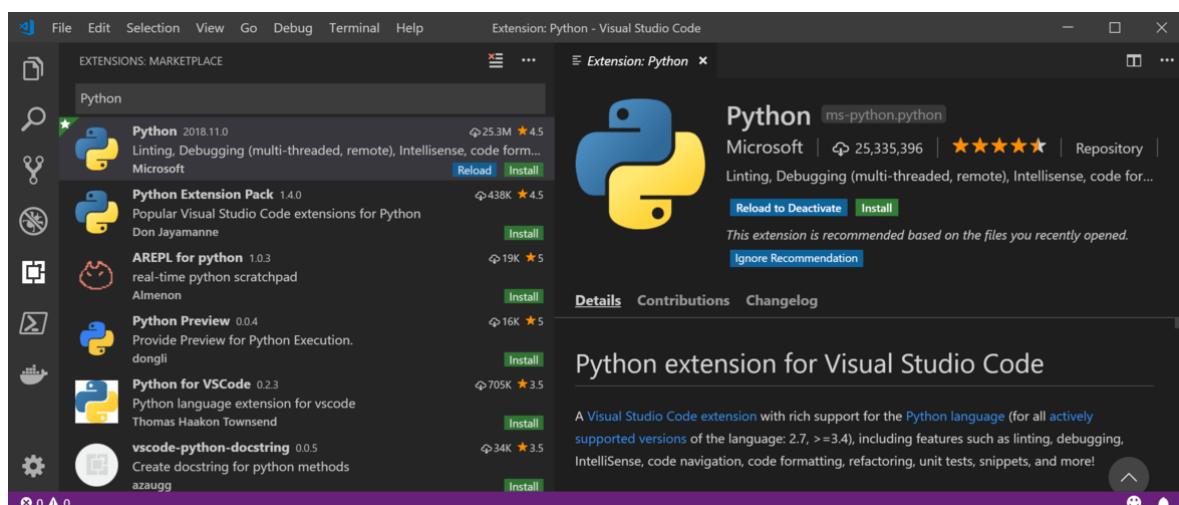
# Starts speech recognition, and returns after a single utterance is recognized. The end of a
# single utterance is determined by listening for silence at the end or until a maximum of 15
# seconds of audio is processed. The task returns the recognition text as result.
# Note: Since recognize_once() returns only a single utterance, it is suitable only for single
# shot recognition like command or query.
# For long-running multi-utterance recognition, use start_continuous_recognition() instead.
result = speech_recognizer.recognize_once()

# Checks result.
if result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("Recognized: {}".format(result.text))
elif result.reason == speechsdk.ResultReason.NoMatch:
    print("No speech could be recognized: {}".format(result.no_match_details))
elif result.reason == speechsdk.ResultReason.Canceled:
    cancellation_details = result.cancellation_details
    print("Speech Recognition canceled: {}".format(cancellation_details.reason))
    if cancellation_details.reason == speechsdk.CancellationReason.Error:
        print("Error details: {}".format(cancellation_details.error_details))

```

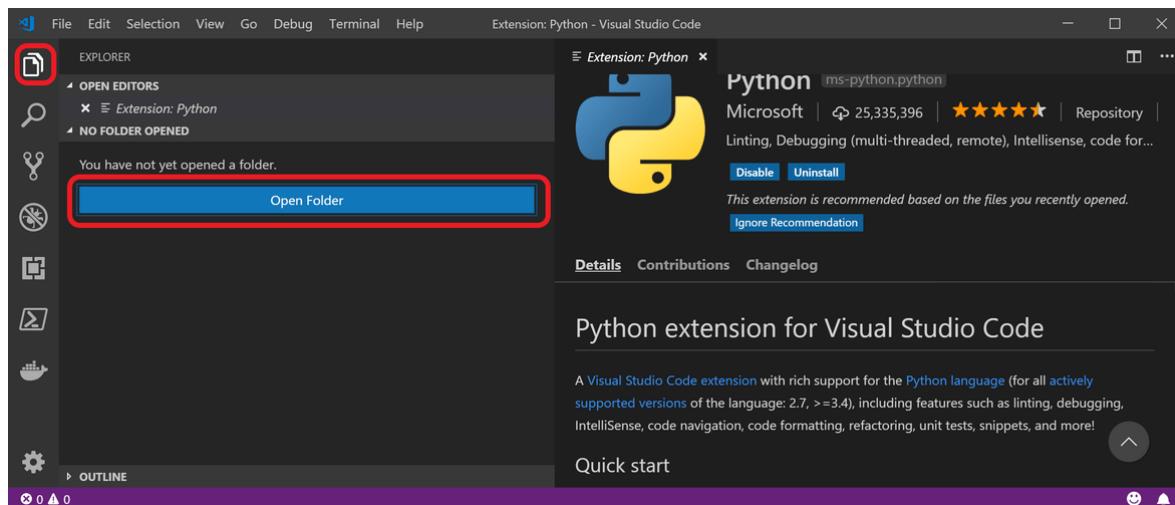
Install and use the Speech SDK with Visual Studio Code

1. Download and install a 64-bit version of [Python](#), 3.5 or later, on your computer.
2. Download and install [Visual Studio Code](#).
3. Open Visual Studio Code and install the Python extension. Select **File > Preferences > Extensions** from the menu. Search for **Python**.

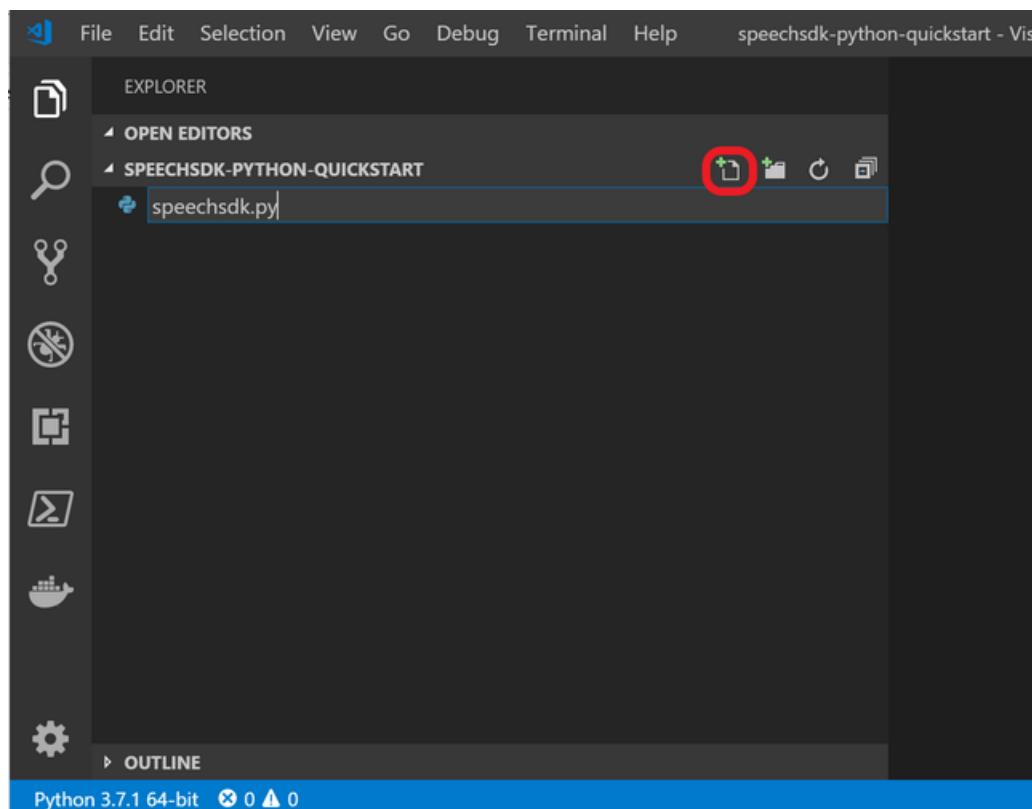


4. Create a folder to store the project in. An example is by using Windows Explorer.

5. In Visual Studio Code, select the **File** icon. Then open the folder you created.



6. Create a new Python source file, `speechsdk.py`, by selecting the new file icon.



7. Copy, paste, and save the **Python code** to the newly created file.
8. Insert your Speech service subscription information.
9. If selected, a Python interpreter displays on the left side of the status bar at the bottom of the window. Otherwise, bring up a list of available Python interpreters. Open the command palette (Ctrl+Shift+P) and enter **Python: Select Interpreter**. Choose an appropriate one.
10. You can install the Speech SDK Python package from within Visual Studio Code. Do that if it's not installed yet for the Python interpreter you selected. To install the Speech SDK package, open a terminal. Bring up the command palette again (Ctrl+Shift+P) and enter **Terminal: Create New Integrated Terminal**. In the terminal that opens, enter the command
`python -m pip install azure-cognitiveservices-speech` or the appropriate command for your system.
11. To run the sample code, right-click somewhere inside the editor. Select **Run Python File in Terminal**. The first 15 seconds of speech input from your audio file will be recognized and logged in the console window.

```
Recognizing first result...
We recognized: What's the weather like?
```

If you have issues following these instructions, refer to the more extensive [Visual Studio Code Python tutorial](#).

Next steps

[Explore Python samples on GitHub](#)

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	ADDITIONAL QUICKSTARTS	CODE SAMPLES
C++		Windows , Linux , macOS
C#		.NET Framework , .NET Core , UWP , Unity , Xamarin
Java		Android , JRE
Javascript		Browser
Node.js		Windows , Linux , macOS
Objective-C	macOs , iOS	iOS , macOS
Python		Windows , Linux , macOS
Swift	macOs , iOS	iOS , macOS

Quickstart: Recognize speech from a microphone

12/10/2019 • 42 minutes to read • [Edit Online](#)

In this quickstart, you'll use the [Speech SDK](#) to interactively recognize speech from a microphone input, and get the text transcription from captured audio. It's easy to integrate this feature into your apps or devices for common recognition tasks, such as transcribing conversations. It can also be used for more complex integrations, like using the Bot Framework with the Speech SDK to build voice assistants.

After satisfying a few prerequisites, recognizing speech from a microphone only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechRecognizer` object using the `SpeechConfig` object from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [.NET](#)
- [Unity](#)
- [UWP](#)
- [Xamarin](#)

Prerequisites

Before you get started:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)
- Make sure that you have access to a microphone for audio capture

Open your project in Visual Studio

The first step is to make sure that you have your project open in Visual Studio.

1. Launch Visual Studio 2019.
2. Load your project and open `Program.cs`.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project. Make note that you've created an `async` method called `RecognizeSpeechAsync()`.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeSpeechAsync()
        {
        }

        static void Main()
        {
            RecognizeSpeechAsync().Wait();
            Console.WriteLine("Please press <Return> to continue.");
            Console.ReadLine();
        }
    }
}
```

Create a Speech configuration

Before you can initialize a `SpeechRecognizer` object, you need to create a configuration that uses your subscription key and subscription region. Insert this code in the `RecognizeSpeechAsync()` method.

NOTE

This sample uses the `FromSubscription()` method to build the `SpeechConfig`. For a full list of available methods, see [SpeechConfig Class](#).

```
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Initialize a SpeechRecognizer

Now, let's create a `SpeechRecognizer`. This object is created inside of a using statement to ensure the proper release of unmanaged resources. Insert this code in the `RecognizeSpeechAsync()` method, right below your Speech configuration.

```
using (var recognizer = new SpeechRecognizer(config))
{}
```

Recognize a phrase

From the `SpeechRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech.

Inside the using statement, add this code:

```
var result = await recognizer.RecognizeOnceAsync();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, you'll want to do something with it. We're going to keep it simple and print the result to console.

Inside the using statement, below `RecognizeOnceAsync()`, add this code:

```
if (result.Reason == ResultReason.RecognizedSpeech)
{
    Console.WriteLine($"We recognized: {result.Text}");
}
else if (result.Reason == ResultReason.NoMatch)
{
    Console.WriteLine($"NOMATCH: Speech could not be recognized.");
}
else if (result.Reason == ResultReason.Canceled)
{
    var cancellation = CancellationDetails.FromResult(result);
    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

    if (cancellation.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }
}
```

Check your code

At this point, your code should look like this:

```

// Copyright (c) Microsoft. All rights reserved.
// Licensed under the MIT license. See LICENSE.md file in the project root for full license
// information.
//

using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeSpeechAsync()
        {
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            using (var recognizer = new SpeechRecognizer(config))
            {
                var result = await recognizer.RecognizeOnceAsync();

                if (result.Reason == ResultReason.RecognizedSpeech)
                {
                    Console.WriteLine($"We recognized: {result.Text}");
                }
                else if (result.Reason == ResultReason.NoMatch)
                {
                    Console.WriteLine($"NOMATCH: Speech could not be recognized.");
                }
                else if (result.Reason == ResultReason.Canceled)
                {
                    var cancellation = CancellationDetails.FromResult(result);
                    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

                    if (cancellation.Reason == CancellationReason.Error)
                    {
                        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                        Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
                        Console.WriteLine($"CANCELED: Did you update the subscription info?");
                    }
                }
            }
        }

        static void Main()
        {
            RecognizeSpeechAsync().Wait();
            Console.WriteLine("Please press <Return> to continue.");
            Console.ReadLine();
        }
    }
}

```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

- Compile the code** - From the menu bar of Visual Studio, choose **Build > Build Solution**.
- Start your app** - From the menu bar, choose **Debug > Start Debugging** or press **F5**.
- Start recognition** - It'll prompt you to speak a phrase in English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

Next steps

[Explore C# samples on GitHub](#)

In this quickstart, you'll use the [Speech SDK](#) to interactively recognize speech from a microphone input, and get the text transcription from captured audio. It's easy to integrate this feature into your apps or devices for common recognition tasks, such as transcribing conversations. It can also be used for more complex integrations, like using the Bot Framework with the Speech SDK to build voice assistants.

After satisfying a few prerequisites, recognizing speech from a microphone only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechRecognizer` object using the `SpeechConfig` object from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [Linux](#)
- [macOS](#)
- [Windows](#)

Prerequisites

Before you get started:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)
- Make sure that you have access to a microphone for audio capture

Add sample code

1. Create a C++ source file named `helloworld.cpp`, and paste the following code into it.

```

#include <iostream> // cin, cout
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;

void recognizeSpeech() {
    // Creates an instance of a speech config with specified subscription key and service
    region.
    // Replace with your own subscription key and service region (e.g., "westus").
    auto config = SpeechConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");

    // Creates a speech recognizer
    auto recognizer = SpeechRecognizer::FromConfig(config);
    cout << "Say something...\n";

    // Starts speech recognition, and returns after a single utterance is recognized. The end
    of a
    // single utterance is determined by listening for silence at the end or until a maximum of
15
    // seconds of audio is processed. The task returns the recognition text as result.
    // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only
for single
    // shot recognition like command or query.
    // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync()
instead.
    auto result = recognizer->RecognizeOnceAsync().get();

    // Checks result.
    if (result->Reason == ResultReason::RecognizedSpeech) {
        cout << "We recognized: " << result->Text << std::endl;
    }
    else if (result->Reason == ResultReason::NoMatch) {
        cout << "NOMATCH: Speech could not be recognized." << std::endl;
    }
    else if (result->Reason == ResultReason::Canceled) {
        auto cancellation = CancellationDetails::FromResult(result);
        cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

        if (cancellation->Reason == CancellationReason::Error) {
            cout << "CANCELED: ErrorCode= " << (int)cancellation->ErrorCode << std::endl;
            cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
            cout << "CANCELED: Did you update the subscription info?" << std::endl;
        }
    }
}

int main(int argc, char **argv) {
    setlocale(LC_ALL, "");
    recognizeSpeech();
    return 0;
}

```

2. In this new file, replace the string `YourSubscriptionKey` with your Speech service subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).

NOTE

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Build the app

NOTE

Make sure to enter the commands below as a *single command line*. The easiest way to do that is to copy the command by using the **Copy** button next to each command, and then paste it at your shell prompt.

- On an **x64** (64-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I  
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core  
-L "$SPEECHSDK_ROOT/lib/x64" -l:libasound.so.2
```

- On an **x86** (32-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I  
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core  
-L "$SPEECHSDK_ROOT/lib/x86" -l:libasound.so.2
```

- On an **ARM64** (64-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I  
"$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core  
-L "$SPEECHSDK_ROOT/lib/arm64" -l:libasound.so.2
```

Run the app

- Configure the loader's library path to point to the Speech SDK library.

- On an **x64** (64-bit) system, enter the following command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/x64"
```

- On an **x86** (32-bit) system, enter this command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/x86"
```

- On an **ARM64** (64-bit) system, enter the following command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/arm64"
```

- Run the application.

```
./helloworld
```

- In the console window, a prompt appears, requesting that you say something. Speak an English phrase or sentence. Your speech is transmitted to the Speech service and transcribed to text, which appears in the same window.

```
Say something...
We recognized: What's the weather like?
```

Next steps

[Explore C++ samples on GitHub](#)

In this quickstart, you'll use the [Speech SDK](#) to interactively recognize speech from a microphone input, and get the text transcription from captured audio. It's easy to integrate this feature into your apps or devices for common recognition tasks, such as transcribing conversations. It can also be used for more complex integrations, like using the Bot Framework with the Speech SDK to build voice assistants.

After satisfying a few prerequisites, recognizing speech from a microphone only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechRecognizer` object using the `SpeechConfig` object from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [Java Runtime](#)
- [Android](#)

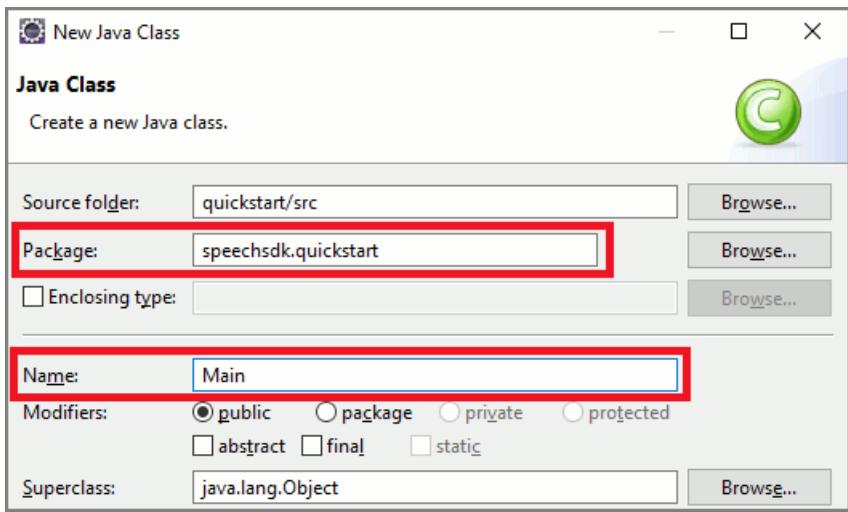
Prerequisites

Before you get started:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)
- Make sure that you have access to a microphone for audio capture

Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in `Main.java` with the following snippet:

```

package speechsdk.quickstart;

import java.util.concurrent.Future;
import com.microsoft.cognitiveservices.speech.*;

/**
 * Quickstart: recognize speech using the Speech SDK for Java.
 */
public class Main {

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {
        try {
            // Replace below with your own subscription key
            String speechSubscriptionKey = "YourSubscriptionKey";
            // Replace below with your own service region (e.g., "westus").
            String serviceRegion = "YourServiceRegion";

            int exitCode = 1;
            SpeechConfig config = SpeechConfig.fromSubscription(speechSubscriptionKey,
serviceRegion);
            assert(config != null);

            SpeechRecognizer reco = new SpeechRecognizer(config);
            assert(reco != null);

            System.out.println("Say something...");

            Future<SpeechRecognitionResult> task = reco.recognizeOnceAsync();
            assert(task != null);

            SpeechRecognitionResult result = task.get();
            assert(result != null);

            if (result.getReason() == ResultReason.RecognizedSpeech) {
                System.out.println("We recognized: " + result.getText());
                exitCode = 0;
            }
            else if (result.getReason() == ResultReason.NoMatch) {
                System.out.println("NOMATCH: Speech could not be recognized.");
            }
            else if (result.getReason() == ResultReason.Canceled) {
                CancellationDetails cancellation = CancellationDetails.fromResult(result);
                System.out.println("CANCELED: Reason=" + cancellation.getReason());

                if (cancellation.getReason() == CancellationReason.Error) {
                    System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
                    System.out.println("CANCELED: ErrorDetails=" +
cancellation.getErrorDetails());
                    System.out.println("CANCELED: Did you update the subscription info?");
                }
            }

            reco.close();

            System.exit(exitCode);
        } catch (Exception ex) {
            System.out.println("Unexpected exception: " + ex.getMessage());

            assert(false);
            System.exit(1);
        }
    }
}

```

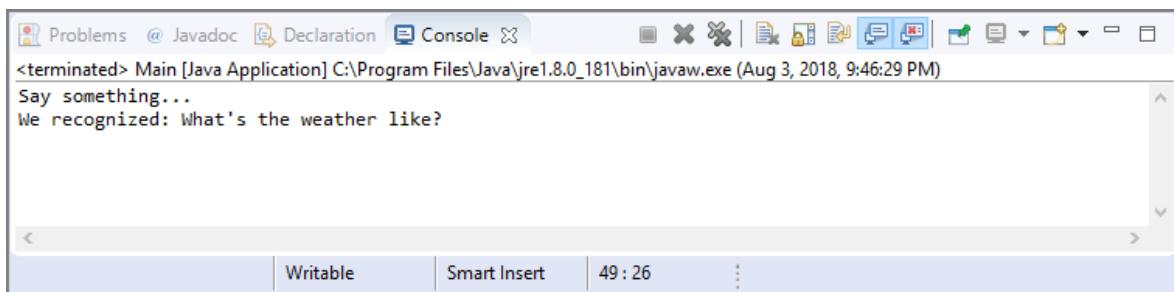
4. Replace the string `YourSubscriptionKey` with your subscription key.
5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
6. Save changes to the project.

NOTE

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Build and run the app

Press F11, or select **Run > Debug**. The next 15 seconds of speech input from your microphone will be recognized and logged in the console window.



Next steps

[Explore Java samples on GitHub](#)

In this quickstart, you'll use the [Speech SDK](#) to interactively recognize speech from a microphone input, and get the text transcription from captured audio. It's easy to integrate this feature into your apps or devices for common recognition tasks, such as transcribing conversations. It can also be used for more complex integrations, like using the Bot Framework with the Speech SDK to build voice assistants.

After satisfying a few prerequisites, recognizing speech from a microphone only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechRecognizer` object using the `SpeechConfig` object from above.
- Using the `SpeechRecognizer` object, start the recognition process for a single utterance.
- Inspect the `SpeechRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)
- Make sure that you have access to a microphone for audio capture

Support and updates

Updates to the Speech SDK Python package are distributed via PyPI and announced in the [Release notes](#).

If a new version is available, you can update to it with the command

```
pip install --upgrade azure-cognitiveservices-speech
```

. Check which version is currently installed by inspecting the `azure.cognitiveservices.speech._version_` variable.

If you have a problem, or you're missing a feature, see [Support and help options](#).

Create a Python application that uses the Speech SDK

Run the sample

You can copy the [sample code](#) from this quickstart to a source file `quickstart.py` and run it in your IDE or in the console:

```
python quickstart.py
```

Or you can download this quickstart tutorial as a [Jupyter](#) notebook from the [Speech SDK sample repository](#) and run it as a notebook.

Sample code

NOTE

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

```
import azure.cognitiveservices.speech as speechsdk

# Creates an instance of a speech config with specified subscription key and service region.
# Replace with your own subscription key and service region (e.g., "westus").
speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"
speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)

# Creates a recognizer with the given settings
speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config)

print("Say something...")

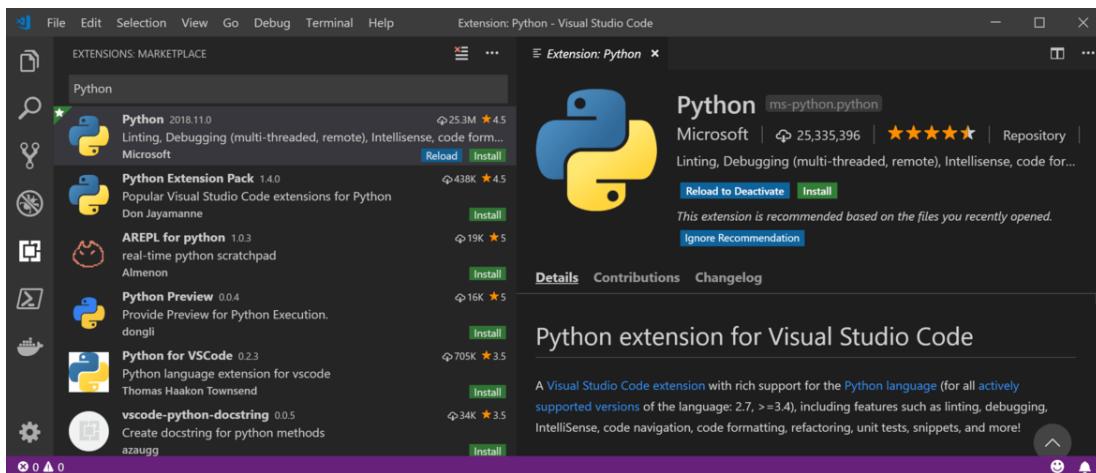
# Starts speech recognition, and returns after a single utterance is recognized. The end of a
# single utterance is determined by listening for silence at the end or until a maximum of 15
# seconds of audio is processed. The task returns the recognition text as result.
# Note: Since recognize_once() returns only a single utterance, it is suitable only for single
# shot recognition like command or query.
# For long-running multi-utterance recognition, use start_continuous_recognition() instead.
result = speech_recognizer.recognize_once()

# Checks result.
if result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("Recognized: {}".format(result.text))
elif result.reason == speechsdk.ResultReason.NoMatch:
    print("No speech could be recognized: {}".format(result.no_match_details))
elif result.reason == speechsdk.ResultReason.Canceled:
    cancellation_details = result.cancellation_details
    print("Speech Recognition canceled: {}".format(cancellation_details.reason))
    if cancellation_details.reason == speechsdk.CancellationReason.Error:
        print("Error details: {}".format(cancellation_details.error_details))
```

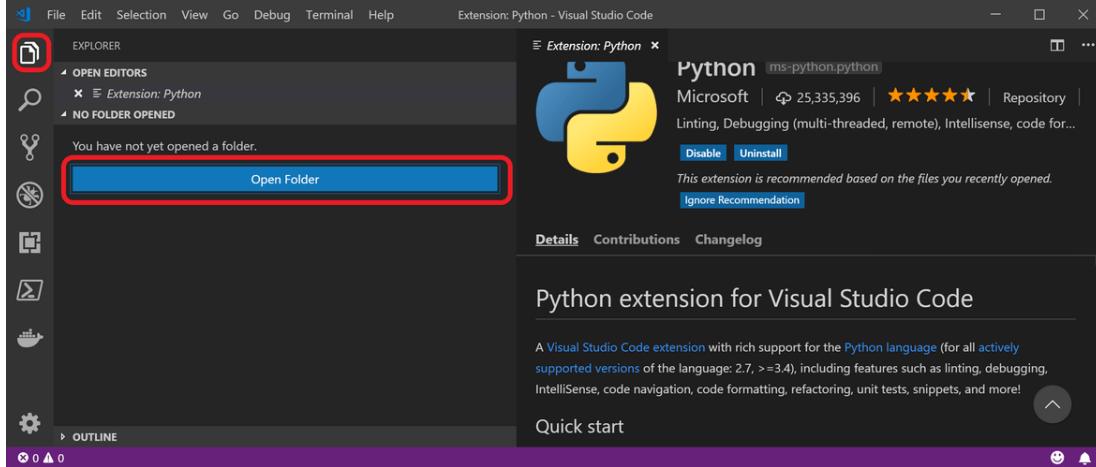
Install and use the Speech SDK with Visual Studio Code

1. Download and install a 64-bit version of [Python](#), 3.5 or later, on your computer.

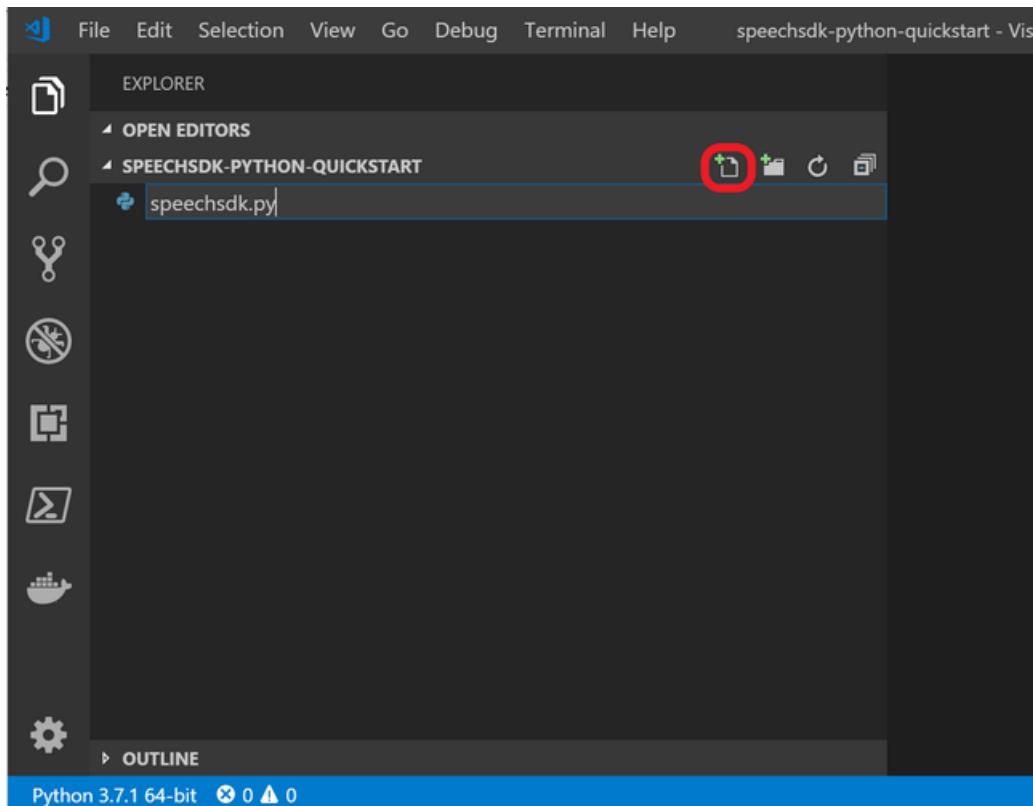
2. Download and install [Visual Studio Code](#).
3. Open Visual Studio Code and install the Python extension. Select **File > Preferences > Extensions** from the menu. Search for **Python**.



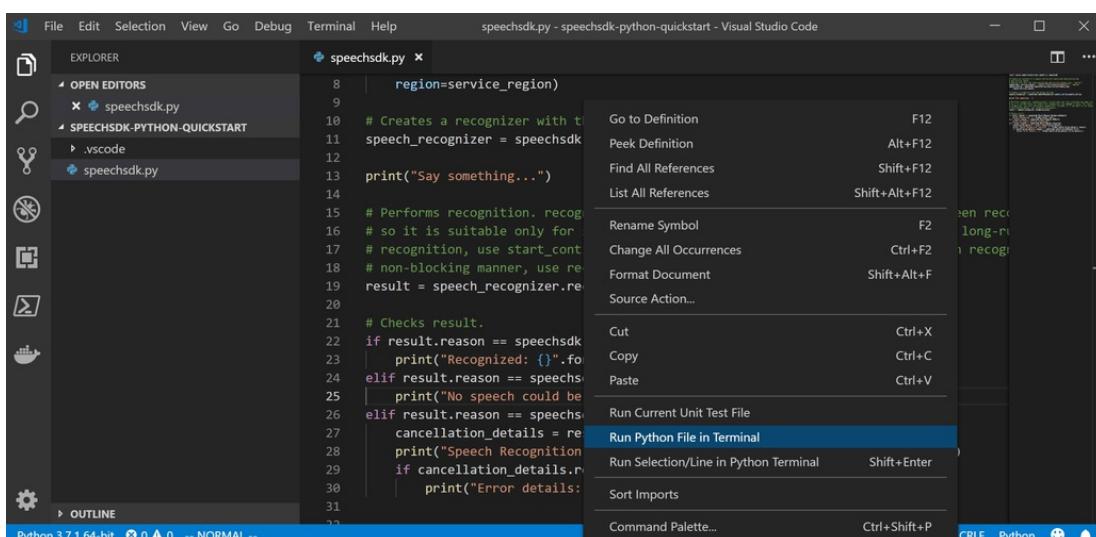
4. Create a folder to store the project in. An example is by using Windows Explorer.
5. In Visual Studio Code, select the **File** icon. Then open the folder you created.



6. Create a new Python source file, `speechsdk.py`, by selecting the new file icon.



7. Copy, paste, and save the [Python code](#) to the newly created file.
8. Insert your Speech service subscription information.
9. If selected, a Python interpreter displays on the left side of the status bar at the bottom of the window. Otherwise, bring up a list of available Python interpreters. Open the command palette (Ctrl+Shift+P) and enter **Python: Select Interpreter**. Choose an appropriate one.
10. You can install the Speech SDK Python package from within Visual Studio Code. Do that if it's not installed yet for the Python interpreter you selected. To install the Speech SDK package, open a terminal. Bring up the command palette again (Ctrl+Shift+P) and enter **Terminal: Create New Integrated Terminal**. In the terminal that opens, enter the command
`python -m pip install azure-cognitiveservices-speech` or the appropriate command for your system.
11. To run the sample code, right-click somewhere inside the editor. Select **Run Python File in Terminal**. Speak a few words when you're prompted. The transcribed text displays shortly afterward.



If you have issues following these instructions, refer to the more extensive [Visual Studio Code Python tutorial](#).

Next steps

[Explore Python samples on GitHub](#)

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	ADDITIONAL QUICKSTARTS	CODE SAMPLES
C++		Windows , Linux , macOS
C#		.NET Framework , .NET Core , UWP , Unity , Xamarin
Java		Android , JRE
Javascript		Browser
Node.js		Windows , Linux , macOS
Objective-C	macOs , iOS	iOS , macOS
Python		Windows , Linux , macOS
Swift	macOs , iOS	iOS , macOS

Quickstart: Recognize speech stored in blob storage

11/6/2019 • 37 minutes to read • [Edit Online](#)

In this quickstart, you will use a REST API to recognize speech from files in a batch process. A batch process executes the speech transcription without any user interactions. It gives you a simple programming model, without the need to manage concurrency, custom speech models, or other details. It entails advanced control options, while making efficient use of Azure speech service resources.

The [batch transcription overview](#) describes the details to use this feature. The detailed API is available as a [Swagger document](#), under the heading `Custom Speech transcriptins`.

The following quickstart will walk you through a usage sample.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Upload a source file to an azure blob](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Open your project in Visual Studio

The first step is to make sure that you have your project open in Visual Studio.

1. Launch Visual Studio 2019.
2. Load your project and open `Program.cs`.

Add a reference to NewtonSoftJSon

1. In the Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show the NuGet Package Manager.
2. In the upper-right corner, find the **Package Source** drop-down box, and make sure that `nuget.org` is selected.
3. In the upper-left corner, select **Browse**.
4. In the search box, type `newtonsoft.json` and select **Enter**.
5. From the search results, select the **Newtonsoft.Json** package, and then select **Install** to install the latest stable version.
6. Accept all agreements and licenses to start the installation.

After the package is installed, a confirmation appears in the **Package Manager Console** window.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project.

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Threading.Tasks;
using System.Net.Http;
using System.Net.Http.Formatting;

namespace BatchClient
{
    class Program
    {
        // Replace with your subscription key
        private const string SubscriptionKey = "YourSubscriptionKey";

        // Update with your service region
        private const string Region = "YourServiceRegion";
        private const int Port = 443;

        // recordings and locale
        private const string Locale = "en-US";
        private const string RecordingsBlobUri = "YourFileUrl";

        //name and description
        private const string Name = "Simple transcription";
        private const string Description = "Simple transcription description";

        private const string speechToTextBasePath = "api/speechtotext/v2.0/";

        static void Main(string[] args)
        {
            TranscribeAsync().Wait();
        }

        static async Task TranscribeAsync()
        {
            Console.WriteLine("Starting transcriptions client...");
        }
    }
}
```

(You'll need to replace the values of `YourSubscriptionKey`, `YourServiceRegion`, and `YourFileUrl` with your own values.)

JSON Wrappers

As the REST API's take requests in JSON format and also return results in JSON we could interact with them using only strings, but that's not recommended. In order to make the requests and responses easier to manage, we'll declare a few classes to use for serializing / deserializing the JSON.

Go ahead and put their declarations after `TranscribeAsync`.

```
public sealed class ModelIdentity
{
    private ModelIdentity(Guid id)
    {
        this.Id = id;
    }

    public Guid Id { get; private set; }

    public static ModelIdentity Create(Guid Id)
```

```

    {
        return new ModelIdentity(Id);
    }
}

public sealed class Transcription
{
    [JsonConstructor]
    private Transcription(Guid id, string name, string description, string locale, DateTime createdDateTime,
    DateTime lastActionDateTime, string status, Uri recordingsUrl, IReadOnlyDictionary<string, string>
    resultsUrls)
    {
        this.Id = id;
        this.Name = name;
        this.Description = description;
        this.CreatedDateTime = createdDateTime;
        this.LastActionDateTime = lastActionDateTime;
        this.Status = status;
        this.Locale = locale;
        this.RecordingsUrl = recordingsUrl;
        this.ResultsUrls = resultsUrls;
    }

    /// <inheritdoc />
    public string Name { get; set; }

    /// <inheritdoc />
    public string Description { get; set; }

    /// <inheritdoc />
    public string Locale { get; set; }

    /// <inheritdoc />
    public Uri RecordingsUrl { get; set; }

    /// <inheritdoc />
    public IReadOnlyDictionary<string, string> ResultsUrls { get; set; }

    public Guid Id { get; set; }

    /// <inheritdoc />
    public DateTime CreatedDateTime { get; set; }

    /// <inheritdoc />
    public DateTime LastActionDateTime { get; set; }

    /// <inheritdoc />
    public string Status { get; set; }

    public string StatusMessage { get; set; }
}

public sealed class TranscriptionDefinition
{
    private TranscriptionDefinition(string name, string description, string locale, Uri recordingsUrl,
    IEnumerable<ModelIdentity> models)
    {
        this.Name = name;
        this.Description = description;
        this.RecordingsUrl = recordingsUrl;
        this.Locale = locale;
        this.Models = models;
        this.properties = new Dictionary<string, string>();
        this.properties.Add("PunctuationMode", "DictatedAndAutomatic");
        this.properties.Add("ProfanityFilterMode", "Masked");
        this.properties.Add("AddWordLevelTimestamps", "True");
    }

    /// <inheritdoc />
}

```

```

    ...
    public string Name { get; set; }

    /// <inheritdoc />
    public string Description { get; set; }

    /// <inheritdoc />
    public Uri RecordingsUrl { get; set; }

    public string Locale { get; set; }

    public IEnumerable<ModelIdentity> Models { get; set; }

    public IDictionary<string, string> properties { get; set; }

    public static TranscriptionDefinition Create(
        string name,
        string description,
        string locale,
        Uri recordingsUrl)
    {
        return TranscriptionDefinition.Create(name, description, locale, recordingsUrl, new
ModelIdentity[0]);
    }

    public static TranscriptionDefinition Create(
        string name,
        string description,
        string locale,
        Uri recordingsUrl,
        IEnumerable<ModelIdentity> models)
    {
        return new TranscriptionDefinition(name, description, locale, recordingsUrl, models);
    }
}

public class AudioFileResult
{
    public string AudioFileName { get; set; }
    public List<SegmentResult> SegmentResults { get; set; }
}

public class RootObject
{
    public List<AudioFileResult> AudioFileResults { get; set; }
}

public class NBest
{
    public double Confidence { get; set; }
    public string Lexical { get; set; }
    public string ITN { get; set; }
    public string MaskedITN { get; set; }
    public string Display { get; set; }
}

public class SegmentResult
{
    public string RecognitionStatus { get; set; }
    public string Offset { get; set; }
    public string Duration { get; set; }
    public List<NBest> NBest { get; set; }
}

```

Create and configure an Http Client

The first thing we'll need is an Http Client that has a correct base URL and authentication set. Insert this code in

```
TranscribeAsync
```

```
var client = new HttpClient();
client.Timeout = TimeSpan.FromMinutes(25);
client.BaseAddress = new UriBuilder(Uri.UriSchemeHttps, $"{Region}.cris.ai", Port).Uri;

client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", SubscriptionKey);
```

Generate a transcription request

Next, we'll generate the transcription request. Add this code to `TranscribeAsync`

```
var path = $"{speechToTextbasePath}Transcriptions/";
var transcriptionDefinition = TranscriptionDefinition.Create(Name, Description, Locale, new
Uri(RecordingsBlobUri));

string res = JsonConvert.SerializeObject(transcriptionDefinition);
StringContent sc = new StringContent(res);
sc.Headers.ContentType = JsonMediaTypeFormatter.DefaultMediaType;
```

Send the request and check its status

Now we post the request to the Speech service and check the initial response code. This response code will simply indicate if the service has received the request. The service will return a Url in the response headers that's the location where it will store the transcription status.

```
Uri transcriptionLocation = null;

using (var response = await client.PostAsync(path, sc))
{
    if (!response.IsSuccessStatusCode)
    {
        Console.WriteLine("Error {0} starting transcription.", response.StatusCode);
        return;
    }

    transcriptionLocation = response.Headers.Location;
}
```

Wait for the transcription to complete

Since the service processes the transcription asynchronously, we need to poll for its status every so often. We'll check every 5 seconds.

We can check the status by retrieving the content at the Url we got when we posted the request. When we get the content back, we deserialize it into one of our helper class to make it easier to interact with.

Here's the polling code with status display for everything except a successful completion, we'll do that next.

```

Console.WriteLine($"Created transcription at location {transcriptionLocation}.");

Console.WriteLine("Checking status.");

bool completed = false;
Transcription transcription = null;

// check for the status of our transcriptions periodically
while (!completed)
{
    // get all transcriptions for the user
    using (var response = await
client.GetAsync(transcriptionLocation.AbsolutePath).ConfigureAwait(false))
    {
        var contentType = response.Content.Headers.ContentType;

        if (response.IsSuccessStatusCode && string.Equals(contentType.MediaType, "application/json",
StringComparison.OrdinalIgnoreCase))
        {
            transcription = await response.Content.ReadAsAsync<Transcription>().ConfigureAwait(false);
        }
        else
        {
            Console.WriteLine("Error with status {0} getting transcription result", response.StatusCode);
            continue;
        }
    }

    // for each transcription in the list we check the status
    switch (transcription.Status)
    {
        case "Failed":
            completed = true;
            Console.WriteLine("Transcription failed. Status: {0}", transcription.StatusMessage);
            break;
        case "Succeeded":
            break;

        case "Running":
            Console.WriteLine("Transcription is still running.");
            break;

        case "NotStarted":
            Console.WriteLine("Transcription has not started.");
            break;
    }

    await Task.Delay(TimeSpan.FromSeconds(5)).ConfigureAwait(false);
}

Console.WriteLine("Press any key...");
Console.ReadKey();
}

```

Display the transcription results

Once the service has successfully completed the transcription the results will be stored in another Url that we can get from the status response.

Here we make a request to download those results in to a temporary file before reading and deserializing them. Once the results are loaded we can print them to the console.

```

completed = true;

var resultsUri0 = transcription.ResultsUrls["channel_0"];

WebClient webClient = new WebClient();

var filename = Path.GetTempFileName();
webClient.DownloadFile(resultsUri0, filename);
var results0 = File.ReadAllText(filename);
var resultObject0 = JsonConvert.DeserializeObject<RootObject>(results0);
Console.WriteLine(results0);

Console.WriteLine("Transcription succeeded. Results: ");
Console.WriteLine(results0);

```

Check your code

At this point, your code should look like this: (We've added some comments to this version)

```

using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Threading.Tasks;
using System.Net.Http;
using System.Net.Http.Formatting;

namespace BatchClient
{
    class Program
    {
        // Replace with your subscription key
        private const string SubscriptionKey = "YourSubscriptionKey";

        // Update with your service region
        private const string Region = "YourServiceRegion";
        private const int Port = 443;

        // recordings and locale
        private const string Locale = "en-US";
        private const string RecordingsBlobUri = "YourFileUrl";

        //name and description
        private const string Name = "Simple transcription";
        private const string Description = "Simple transcription description";

        private const string speechToTextbasePath = "api/speechtotext/v2.0/";

        static void Main(string[] args)
        {
            TranscribeAsync().Wait();
        }

        static async Task TranscribeAsync()
        {
            Console.WriteLine("Starting transcriptions client...");

            // create the client object and authenticate
            var client = new HttpClient();
            client.Timeout = TimeSpan.FromMinutes(25);
            client.BaseAddress = new UriBuilder(Uri.UriSchemeHttps, $"{Region}.cris.ai", Port).Uri;

            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", SubscriptionKey);

```

```

        var path = $"{speechToTextBasePath}Transcriptions/";
        var transcriptionDefinition = TranscriptionDefinition.Create(Name, Description, Locale, new
Uri(RecordingsBlobUri));

        string res = JsonConvert.SerializeObject(transcriptionDefinition);
        StringContent sc = new StringContent(res);
        sc.Headers.ContentType = JsonMediaTypeFormatter.DefaultMediaType;

        Uri transcriptionLocation = null;

        using (var response = await client.PostAsync(path, sc))
{
    if (!response.IsSuccessStatusCode)
    {
        Console.WriteLine("Error {0} starting transcription.", response.StatusCode);
        return;
    }

    transcriptionLocation = response.Headers.Location;
}

Console.WriteLine($"Created transcription at location {transcriptionLocation}.");
```

Console.WriteLine("Checking status.");

```

bool completed = false;
Transcription transcription = null;

// check for the status of our transcriptions periodically
while (!completed)
{
    // get all transcriptions for the user
    using (var response = await
client.GetAsync(transcriptionLocation.AbsolutePath).ConfigureAwait(false))
    {
        var contentType = response.Content.Headers.ContentType;

        if (response.IsSuccessStatusCode && string.Equals(contentType.MediaType,
"application/json", StringComparison.OrdinalIgnoreCase))
        {
            transcription = await response.Content.ReadAsAsync<Transcription>
().ConfigureAwait(false);
        }
        else
        {
            Console.WriteLine("Error with status {0} getting transcription result",
response.StatusCode);
            continue;
        }
    }

    // for each transcription in the list we check the status
    switch (transcription.Status)
    {
        case "Failed":
            completed = true;
            Console.WriteLine("Transcription failed. Status: {0}", transcription.StatusMessage);
            break;
        case "Succeeded":
            completed = true;

            var resultsUri0 = transcription.ResultsUrls["channel_0"];

            WebClient webClient = new WebClient();

            var filename = Path.GetTempFileName();
            webClient.DownloadFile(resultsUri0, filename);
            var results0 = File.ReadAllText(filename);
```

```

        var resultObject0 = JsonConvert.DeserializeObject<RootObject>(results0);
        Console.WriteLine(results0);

        Console.WriteLine("Transcription succeeded. Results: ");
        Console.WriteLine(results0);
        break;

        case "Running":
            Console.WriteLine("Transcription is still running.");
            break;

        case "NotStarted":
            Console.WriteLine("Transcription has not started.");
            break;
    }

    await Task.Delay(TimeSpan.FromSeconds(5)).ConfigureAwait(false);
}

Console.WriteLine("Press any key...");
Console.ReadKey();
}

public sealed class ModelIdentity
{
    private ModelIdentity(Guid id)
    {
        this.Id = id;
    }

    public Guid Id { get; private set; }

    public static ModelIdentity Create(Guid Id)
    {
        return new ModelIdentity(Id);
    }
}

public sealed class Transcription
{
    [JsonConstructor]
    private Transcription(Guid id, string name, string description, string locale, DateTime
createdDateTime, DateTime lastActionDateTime, string status, Uri recordingsUrl, IReadOnlyDictionary<string,
string> resultsUrls)
    {
        this.Id = id;
        this.Name = name;
        this.Description = description;
        this.CreatedDateTime = createdDateTime;
        this.LastActionDateTime = lastActionDateTime;
        this.Status = status;
        this.Locale = locale;
        this.RecordingsUrl = recordingsUrl;
        this.ResultsUrls = resultsUrls;
    }

    /// <inheritdoc />
    public string Name { get; set; }

    /// <inheritdoc />
    public string Description { get; set; }

    /// <inheritdoc />
    public string Locale { get; set; }

    /// <inheritdoc />
    public Uri RecordingsUrl { get; set; }
}

```

```

/// <inheritdoc />
public IReadOnlyDictionary<string, string> ResultsUrls { get; set; }

public Guid Id { get; set; }

/// <inheritdoc />
public DateTime CreatedDateTime { get; set; }

/// <inheritdoc />
public DateTime LastActionDateTime { get; set; }

/// <inheritdoc />
public string Status { get; set; }

public string StatusMessage { get; set; }
}

public sealed class TranscriptionDefinition
{
    private TranscriptionDefinition(string name, string description, string locale, Uri recordingsUrl,
IEnumerable<ModelIdentity> models)
    {
        this.Name = name;
        this.Description = description;
        this.RecordingsUrl = recordingsUrl;
        this.Locale = locale;
        this.Models = models;
        this.properties = new Dictionary<string, string>();
        this.properties.Add("PunctuationMode", "DictatedAndAutomatic");
        this.properties.Add("ProfanityFilterMode", "Masked");
        this.properties.Add("AddWordLevelTimestamps", "True");
    }

    /// <inheritdoc />
    public string Name { get; set; }

    /// <inheritdoc />
    public string Description { get; set; }

    /// <inheritdoc />
    public Uri RecordingsUrl { get; set; }

    public string Locale { get; set; }

    public IEnumerable<ModelIdentity> Models { get; set; }

    public IDictionary<string, string> properties { get; set; }

    public static TranscriptionDefinition Create(
        string name,
        string description,
        string locale,
        Uri recordingsUrl)
    {
        return TranscriptionDefinition.Create(name, description, locale, recordingsUrl, new
ModelIdentity[0]);
    }

    public static TranscriptionDefinition Create(
        string name,
        string description,
        string locale,
        Uri recordingsUrl,
        IEnumerable<ModelIdentity> models)
    {
        return new TranscriptionDefinition(name, description, locale, recordingsUrl, models);
    }
}

```

```

public class AudioFileResult
{
    public string AudioFileName { get; set; }
    public List<SegmentResult> SegmentResults { get; set; }
}

public class RootObject
{
    public List<AudioFileResult> AudioFileResults { get; set; }
}

public class NBest
{
    public double Confidence { get; set; }
    public string Lexical { get; set; }
    public string ITN { get; set; }
    public string MaskedITN { get; set; }
    public string Display { get; set; }
}

public class SegmentResult
{
    public string RecognitionStatus { get; set; }
    public string Offset { get; set; }
    public string Duration { get; set; }
    public List<NBest> NBest { get; set; }
}
}
}

```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

- Compile the code** - From the menu bar of Visual Studio, choose **Build > Build Solution**.
- Start your app** - From the menu bar, choose **Debug > Start Debugging** or press **F5**.
- Start recognition** - It'll prompt you to speak a phrase in English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

Next steps

[Explore C# samples on GitHub](#)

In this quickstart, you will use a REST API to recognize speech from files in a batch process. A batch process executes the speech transcription without any user interactions. It gives you a simple programming model, without the need to manage concurrency, custom speech models, or other details. It entails advanced control options, while making efficient use of Azure speech service resources.

The [batch transcription overview](#) describes the details to use this feature. The detailed API is available as a [Swagger document](#), under the heading `Custom Speech transcriptins`.

The following quickstart will walk you through a usage sample.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)

- [Upload a source file to an azure blob](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Open your project in Visual Studio

The first step is to make sure that you have your project open in Visual Studio.

1. Launch Visual Studio 2019.
2. Load your project and open `helloworld.cpp`.

Add a references

To speed up our code development we'll be using a couple of external components:

- [CPP Rest SDK](#) A client library for making REST calls to a REST service.
- [nlohmann/json](#) Handy JSON Parsing / Serialization / Deserialization library.

Both can be installed using [vcpkg](#).

```
vcpkg install cpprestsdk cpprestsdk:x64-windows  
vcpkg install nlohmann-json
```

Start with some boilerplate code

Let's add some code that works as a skeleton for our project.

```

#include <iostream>
#include <sstream>
#include <Windows.h>
#include <locale>
#include <codecvt>
#include <string>

#include <cpprest/http_client.h>
#include <cpprest/filestream.h>
#include <nlohmann/json.hpp>

using namespace std;
using namespace utility; // Common utilities like string conversions
using namespace web; // Common features like URIs.
using namespace web::http; // Common HTTP functionality
using namespace web::http::client; // HTTP client features
using namespace concurrency::streams; // Asynchronous streams
using json = nlohmann::json;

const string_t region = U("YourServiceRegion");
const string_t subscriptionKey = U("YourSubscriptionKey");
const string name = "Simple transcription";
const string description = "Simple transcription description";
const string myLocale = "en-US";
const string recordingsBlobUri = "YourFileUrl";

void recognizeSpeech()
{
    std::wstring_convert<std::codecvt_utf8_utf16<wchar_t>> converter;
}

int wmain()
{
    recognizeSpeech();
    cout << "Please press a key to continue.\n";
    cin.get();
    return 0;
}

```

(You'll need to replace the values of `YourSubscriptionKey`, `YourServiceRegion`, and `YourFileUrl` with your own values.)

JSON Wrappers

As the REST API's take requests in JSON format and also return results in JSON we could interact with them using only strings, but that's not recommended. In order to make the requests and responses easier to manage, we'll declare a few classes to use for serializing / deserializing the JSON and some methods to assist `nlohmann/json`.

Go ahead and put their declarations before `recognizeSpeech` .

```

class TranscriptionDefinition {
private:
    TranscriptionDefinition(string name,
                           string description,
                           string locale,
                           string recordingsUrl,
                           std::list<string> models) {

        Name = name;
        Description = description;
        RecordingsUrl = recordingsUrl;
    }
}
```

```

        recordingsUrl = recordingsUrl,
        Locale = locale;
        Models = models;
    }

public:
    string Name;
    string Description;
    string RecordingsUrl;
    string Locale;
    std::list<string> Models;
    std::map<string, string> properties;

    static TranscriptionDefinition Create(string name, string description, string locale, string
recordingsUrl) {
        return TranscriptionDefinition(name, description, locale, recordingsUrl, std::list<string>());
    }
    static TranscriptionDefinition Create(string name, string description, string locale, string
recordingsUrl,
        std::list<string> models) {
        return TranscriptionDefinition(name, description, locale, recordingsUrl, models);
    }
};

void to_json(nlohmann::json& j, const TranscriptionDefinition& t) {
    j = nlohmann::json{
        { "description", t.Description },
        { "locale", t.Locale },
        { "models", t.Models },
        { "name", t.Name },
        { "properties", t.properties },
        { "recordingsurl",t.RecordingsUrl }
    };
}

void from_json(const nlohmann::json& j, TranscriptionDefinition& t) {
    j.at("locale").get_to(t.Locale);
    j.at("models").get_to(t.Models);
    j.at("name").get_to(t.Name);
    j.at("properties").get_to(t.properties);
    j.at("recordingsurl").get_to(t.RecordingsUrl);
}

class Transcription {
public:
    string name;
    string description;
    string locale;
    string recordingsUrl;
    map<string, string> resultsUrls;
    string id;
    string createdDateTime;
    string lastActionDateTime;
    string status;
    string statusMessage;
};

void to_json(nlohmann::json& j, const Transcription& t) {
    j = nlohmann::json{
        { "description", t.description },
        { "locale", t.locale },
        { "createddatetime", t.createdDateTime },
        { "name", t.name },
        { "id", t.id },
        { "recordingsurl",t.recordingsUrl },
        { "resultUrls", t.resultsUrls},
        { "status", t.status},
        { "statusmessage", t.statusMessage}
    };
}

```

```

};

void from_json(const nlohmann::json& j, Transcription& t) {
    j.at("description").get_to(t.description);
    j.at("locale").get_to(t.locale);
    j.at("createdDateTime").get_to(t.createdDateTime);
    j.at("name").get_to(t.name);
    j.at("recordingsUrl").get_to(t.recordingsUrl);
    t.resultsUrls = j.at("resultsUrls").get<map<string, string>>();
    j.at("status").get_to(t.status);
    j.at("statusMessage").get_to(t.statusMessage);
}
class Result
{
public:
    string Lexical;
    string ITN;
    string MaskedITN;
    string Display;
};

void from_json(const nlohmann::json& j, Result& r) {
    j.at("Lexical").get_to(r.Lexical);
    j.at("ITN").get_to(r.ITN);
    j.at("MaskedITN").get_to(r.MaskedITN);
    j.at("Display").get_to(r.Display);
}

class NBest : public Result
{
public:
    double Confidence;
};

void from_json(const nlohmann::json& j, NBest& nb) {
    j.at("Confidence").get_to(nb.Confidence);
    j.at("Lexical").get_to(nb.Lexical);
    j.at("ITN").get_to(nb.ITN);
    j.at("MaskedITN").get_to(nb.MaskedITN);
    j.at("Display").get_to(nb.Display);
}

class SegmentResult
{
public:
    string RecognitionStatus;
    ULONG Offset;
    ULONG Duration;
    std::list<NBest> NBest;
};

void from_json(const nlohmann::json& j, SegmentResult& sr) {
    j.at("RecognitionStatus").get_to(sr.RecognitionStatus);
    j.at("Offset").get_to(sr.Offset);
    j.at("Duration").get_to(sr.Duration);
    sr.NBest = j.at("NBest").get<list<NBest>>();
}

class AudioFileResult
{
public:
    string AudioFileName;
    std::list<SegmentResult> SegmentResults;
    std::list<Result> CombinedResults;
};

void from_json(const nlohmann::json& j, AudioFileResult& arf) {
    j.at("AudioFileName").get_to(arf.AudioFileName);
    arf.SegmentResults = j.at("SegmentResults").get<list<SegmentResult>>();
    arf.CombinedResults = j.at("CombinedResults").get<list<Result>>();
}

class RootObject {
    ...
}

```

```

public:
    std::list<AudioFileResult> AudioFileResults;
};

void from_json(const nlohmann::json& j, RootObject& r) {
    r.AudioFileResults = j.at("AudioFileResults").get<list<AudioFileResult>>();
}

```

Create and configure an Http Client

The first thing we'll need is an Http Client that has a correct base URL and authentication set. Insert this code in `recognizeSpeech`

```

utility::string_t service_url = U("https://") + region + U(".cris.ai/api/speechtotext/v2.0/Transcriptions/");
uri u(service_url);

http_client c(u);
http_request msg(methods::POST);
msg.headers().add(U("Content-Type"), U("application/json"));
msg.headers().add(U("Ocp-Apim-Subscription-Key"), subscriptionKey);

```

Generate a transcription request

Next, we'll generate the transcription request. Add this code to `recognizeSpeech`

```

auto transportdef = TranscriptionDefinition::Create(name, description, myLocale, recordingsBlobUri);

nlohmann::json transportdefJSON = transportdef;

msg.set_body(transportdefJSON.dump());

```

Send the request and check its status

Now we post the request to the Speech service and check the initial response code. This response code will simply indicate if the service has received the request. The service will return a Url in the response headers that's the location where it will store the transcription status.

```

auto response = c.request(msg).get();
auto statusCode = response.status_code();

if (statusCode != status_codes::Accepted)
{
    cout << "Unexpected status code " << statusCode << endl;
    return;
}

string_t transcriptionLocation = response.headers()[U("location")];

cout << "Transcription status is located at " << converter.to_bytes(transcriptionLocation) << endl;

```

Wait for the transcription to complete

Since the service processes the transcription asynchronously, we need to poll for its status every so often. We'll check every 5 seconds.

We can check the status by retrieving the content at the Url we got when we posted the request. When we get

the content back, we deserialize it into one of our helper class to make it easier to interact with.

Here's the polling code with status display for everything except a successful completion, we'll do that next.

```
bool completed = false;

while (!completed)
{
    auto statusResponse = statusCheckClient.request(statusCheckMessage).get();
    auto statusResponseCode = statusResponse.status_code();

    if (statusResponseCode != status_codes::OK)
    {
        cout << "Fetching the transcription returned unexpected http code " << statusResponseCode << endl;
        return;
    }

    auto body = statusResponse.extract_string().get();
    nlohmann::json statusJSON = nlohmann::json::parse(body);
    Transcription transcriptionStatus = statusJSON;

    if (!_strcmp(transcriptionStatus.status.c_str(), "Failed"))
    {
        completed = true;
        cout << "Transcription has failed " << transcriptionStatus.statusMessage << endl;
    }
    else if (!_strcmp(transcriptionStatus.status.c_str(), "Succeeded"))
    {
    }
    else if (!_strcmp(transcriptionStatus.status.c_str(), "Running"))
    {
        cout << "Transcription is running." << endl;
    }
    else if (!_strcmp(transcriptionStatus.status.c_str(), "NotStarted"))
    {
        cout << "Transcription has not started." << endl;
    }

    if (!completed) {
        Sleep(5000);
    }
}
```

Display the transcription results

Once the service has successfully completed the transcription the results will be stored in another URL that we can get from the status response.

We'll download the contents of that URL, deserialize the JSON, and loop through the results printing out the display text as we go.

```

completed = true;
cout << "Success!" << endl;
string result = transcriptionStatus.resultsUrls["channel_0"];
cout << "Transcription has completed. Results are at " << result << endl;
cout << "Fetching results" << endl;

http_client result_client(converter.from_bytes(result));
http_request resultMessage(methods::GET);
resultMessage.headers().add(U("Ocp-Apim-Subscription-Key"), subscriptionKey);

auto resultResponse = result_client.request(resultMessage).get();

auto responseCode = resultResponse.status_code();

if (responseCode != status_codes::OK)
{
    cout << "Fetching the transcription returned unexpected http code " << responseCode << endl;
    return;
}

auto resultBody = resultResponse.extract_string().get();

nlohmann::json resultJSON = nlohmann::json::parse(resultBody);
RootObject root = resultJSON;

for (AudioFileResult af : root.AudioFileResults)
{
    cout << "There were " << af.SegmentResults.size() << " results in " << af.AudioFileName << endl;

    for (SegmentResult segResult : af.SegmentResults)
    {
        cout << "Status: " << segResult.RecognitionStatus << endl;

        if (!_stricmp(segResult.RecognitionStatus.c_str(), "success") && segResult.NBest.size() > 0)
        {
            cout << "Best text result was: '" << segResult.NBest.front().Display << "'" << endl;
        }
    }
}

```

Check your code

At this point, your code should look like this: (We've added some comments to this version)

```

#include <iostream>
#include <sstream>
#include <Windows.h>
#include <locale>
#include <codecvt>
#include <string>

#include <cpprest/http_client.h>
#include <cpprest/filestream.h>
#include <nlohmann/json.hpp>

using namespace std; // Common utilities like string conversions
using namespace utility; // Common features like URIs.
using namespace web; // Common HTTP functionality
using namespace web::http; // HTTP client features
using namespace concurrency::streams; // Asynchronous streams
using json = nlohmann::json;

const string_t region = U("YourServiceRegion");
const string_t subscriptionKey = U("YourSubscriptionKey");

```

```

const string name = "Simple transcription";
const string description = "Simple transcription description";
const string myLocale = "en-US";
const string recordingsBlobUri = "YourFileUrl";

class TranscriptionDefinition {
private:
    TranscriptionDefinition(string name,
                           string description,
                           string locale,
                           string recordingsUrl,
                           std::list<string> models) {

        Name = name;
        Description = description;
        RecordingsUrl = recordingsUrl;
        Locale = locale;
        Models = models;
    }

public:
    string Name;
    string Description;
    string RecordingsUrl;
    string Locale;
    std::list<string> Models;
    std::map<string, string> properties;

    static TranscriptionDefinition Create(string name, string description, string locale, string
recordingsUrl) {
        return TranscriptionDefinition(name, description, locale, recordingsUrl, std::list<string>());
    }
    static TranscriptionDefinition Create(string name, string description, string locale, string
recordingsUrl,
        std::list<string> models) {
        return TranscriptionDefinition(name, description, locale, recordingsUrl, models);
    }
};

void to_json(nlohmann::json& j, const TranscriptionDefinition& t) {
    j = nlohmann::json{
        { "description", t.Description },
        { "locale", t.Locale },
        { "models", t.Models },
        { "name", t.Name },
        { "properties", t.properties },
        { "recordingsurl", t.RecordingsUrl }
    };
}

void from_json(const nlohmann::json& j, TranscriptionDefinition& t) {
    j.at("locale").get_to(t.Locale);
    j.at("models").get_to(t.Models);
    j.at("name").get_to(t.Name);
    j.at("properties").get_to(t.properties);
    j.at("recordingsurl").get_to(t.RecordingsUrl);
}

class Transcription {
public:
    string name;
    string description;
    string locale;
    string recordingsUrl;
    map<string, string> resultsUrls;
    string id;
    string createdDateTime;
    string lastActionDateTime;
    string status;
}

```

```

        string statusMessage;
    };

    void to_json(nlohmann::json& j, const Transcription& t) {
        j = nlohmann::json{
            { "description", t.description },
            { "locale", t.locale },
            { "createddatetime", t.dateTime },
            { "name", t.name },
            { "id", t.id },
            { "recordingsurl", t.recordingsUrl },
            { "resultUrls", t.resultsUrls },
            { "status", t.status },
            { "statusmessage", t.statusMessage }
        };
    };

    void from_json(const nlohmann::json& j, Transcription& t) {
        j.at("description").get_to(t.description);
        j.at("locale").get_to(t.locale);
        j.at("createdDateTime").get_to(t.dateTime);
        j.at("name").get_to(t.name);
        j.at("recordingsUrl").get_to(t.recordingsUrl);
        t.resultsUrls = j.at("resultsUrls").get<map<string, string>>();
        j.at("status").get_to(t.status);
        j.at("statusMessage").get_to(t.statusMessage);
    }

    class Result
    {
    public:
        string Lexical;
        string ITN;
        string MaskedITN;
        string Display;
    };
    void from_json(const nlohmann::json& j, Result& r) {
        j.at("Lexical").get_to(r.Lexical);
        j.at("ITN").get_to(r.ITN);
        j.at("MaskedITN").get_to(r.MaskedITN);
        j.at("Display").get_to(r.Display);
    }

    class NBest : public Result
    {
    public:
        double Confidence;
    };
    void from_json(const nlohmann::json& j, NBest& nb) {
        j.at("Confidence").get_to(nb.Confidence);
        j.at("Lexical").get_to(nb.Lexical);
        j.at("ITN").get_to(nb.ITN);
        j.at("MaskedITN").get_to(nb.MaskedITN);
        j.at("Display").get_to(nb.Display);
    }

    class SegmentResult
    {
    public:
        string RecognitionStatus;
        ULONG Offset;
        ULONG Duration;
        std::list<NBest> NBest;
    };
    void from_json(const nlohmann::json& j, SegmentResult& sr) {
        j.at("RecognitionStatus").get_to(sr.RecognitionStatus);
        j.at("Offset").get_to(sr.Offset);
        j.at("Duration").get_to(sr.Duration);
        sr.NBest = j.at("NBest").get<list<NBest>>();
    }
}

```

```

        }

    class AudioFileResult
    {
    public:
        string AudioFileName;
        std::list<SegmentResult> SegmentResults;
        std::list<Result> CombinedResults;
    };
    void from_json(const nlohmann::json& j, AudioFileResult& arf) {
        j.at("AudioFileName").get_to(arf.AudioFileName);
        arf.SegmentResults = j.at("SegmentResults").get<list<SegmentResult>>();
        arf.CombinedResults = j.at("CombinedResults").get<list<Result>>();
    }

    class RootObject {
    public:
        std::list<AudioFileResult> AudioFileResults;
    };
    void from_json(const nlohmann::json& j, RootObject& r) {
        r.AudioFileResults = j.at("AudioFileResults").get<list<AudioFileResult>>();
    }

    void recognizeSpeech()
    {
        std::wstring_convert<std::codecvt_utf8_utf16<wchar_t > > converter;

        utility::string_t service_url = U("https://") + region +
U(".cris.ai/api/speechtotext/v2.0/Transcriptions/");
        uri u(service_url);

        http_client c(u);
        http_request msg(methods::POST);
        msg.headers().add(U("Content-Type"), U("application/json"));
        msg.headers().add(U("Ocp-Apim-Subscription-Key"), subscriptionKey);

        auto transportdef = TranscriptionDefinition::Create(name, description, myLocale, recordingsBlobUri);

        nlohmann::json transportdefJSON = transportdef;

        msg.set_body(transportdefJSON.dump());

        auto response = c.request(msg).get();
        auto statusCode = response.status_code();

        if (statusCode != status_codes::Accepted)
        {
            cout << "Unexpected status code " << statusCode << endl;
            return;
        }

        string_t transcriptionLocation = response.headers()[U("location")];

        cout << "Transcription status is located at " << converter.to_bytes(transcriptionLocation) << endl;

        http_client statusCheckClient(transcriptionLocation);
        http_request statusCheckMessage(methods::GET);
        statusCheckMessage.headers().add(U("Ocp-Apim-Subscription-Key"), subscriptionKey);

        bool completed = false;

        while (!completed)
        {
            auto statusResponse = statusCheckClient.request(statusCheckMessage).get();
            auto statusResponseCode = statusResponse.status_code();

            if (statusResponseCode != status_codes::OK)
            {
                cout << "Fetching the transcription returned unsupported http code " << statusResponseCode <<

```

```

cout << " Fetching the transcription returned unexpected http code " << statusResponseCode <<
endl;
return;
}

auto body = statusResponse.extract_string().get();
nlohmann::json statusJSON = nlohmann::json::parse(body);
Transcription transcriptionStatus = statusJSON;

if (!_strcmp(transcriptionStatus.status.c_str(), "Failed"))
{
    completed = true;
    cout << "Transcription has failed " << transcriptionStatus.statusMessage << endl;
}
else if (!_strcmp(transcriptionStatus.status.c_str(), "Succeeded"))
{
    completed = true;
    cout << "Success!" << endl;
    string result = transcriptionStatus.resultsUrls["channel_0"];
    cout << "Transcription has completed. Results are at " << result << endl;
    cout << "Fetching results" << endl;

    http_client result_client(converter.from_bytes(result));
    http_request resultMessage(methods::GET);
    resultMessage.headers().add(U("Ocp-Apim-Subscription-Key"), subscriptionKey);

    auto resultResponse = result_client.request(resultMessage).get();

    auto responseCode = resultResponse.status_code();

    if (responseCode != status_codes::OK)
    {
        cout << "Fetching the transcription returned unexpected http code " << responseCode << endl;
        return;
    }

    auto responseBody = resultResponse.extract_string().get();

    nlohmann::json resultJSON = nlohmann::json::parse(responseBody);
    RootObject root = resultJSON;

    for (AudioFileResult af : root.AudioFileResults)
    {
        cout << "There were " << af.SegmentResults.size() << " results in " << af.AudioFileName <<
endl;

        for (SegmentResult segResult : af.SegmentResults)
        {
            cout << "Status: " << segResult.RecognitionStatus << endl;

            if (!_strcmp(segResult.RecognitionStatus.c_str(), "success") && segResult.NBest.size() >
0)
            {
                cout << "Best text result was: '" << segResult.NBest.front().Display << "'" << endl;
            }
        }
    }
    else if (!_strcmp(transcriptionStatus.status.c_str(), "Running"))
    {
        cout << "Transcription is running." << endl;
    }
    else if (!_strcmp(transcriptionStatus.status.c_str(), "NotStarted"))
    {
        cout << "Transcription has not started." << endl;
    }
}

if (!completed) {
    Sleep(5000);
}

```

```
    }

}

int wmain()
{
    recognizeSpeech();
    cout << "Please press a key to continue.\n";
    cin.get();
    return 0;
}
```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

Next steps

[Explore C++ samples on GitHub](#)

In this quickstart, you will use a REST API to recognize speech from files in a batch process. A batch process executes the speech transcription without any user interactions. It gives you a simple programming model, without the need to manage concurrency, custom speech models, or other details. It entails advanced control options, while making efficient use of Azure speech service resources.

The [batch transcription overview](#) describes the details to use this feature. The detailed API is available as a [Swagger document](#), under the heading `Custom Speech transcriptins`.

The following quickstart will walk you through a usage sample.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Upload a source file to an azure blob](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Open your project in Eclipse

The first step is to make sure that you have your project open in Eclipse.

1. Launch Eclipse
2. Load your project and open `Main.java`.

Add a reference to Gson

We'll be using an external JSON serializer / deserializer in this quickstart. For Java we've chosen [Gson](#).

Open your pom.xml and add the following reference:

```
<dependencies>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>
</dependencies>
```

Start with some boilerplate code

Let's add some code that works as a skeleton for our project.

```
package quickstart;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Date;
import java.util.Dictionary;
import java.util.Hashtable;
import java.util.UUID;

import com.google.gson.Gson;
public class Main {

    private static String region = "YourServiceRegion";
    private static String subscriptionKey = "YourSubscriptionKey";
    private static String Locale = "en-US";
    private static String RecordingsBlobUri = "YourFileUrl";
    private static String Name = "Simple transcription";
    private static String Description = "Simple transcription description";

    public static void main(String[] args) throws IOException, InterruptedException {
        System.out.println("Starting transcriptions client...");
    }
}
```

(You'll need to replace the values of `YourSubscriptionKey`, `YourServiceRegion`, and `YourFileUrl` with your own values.)

JSON Wrappers

As the REST API's take requests in JSON format and also return results in JSON we could interact with them using only strings, but that's not recommended. In order to make the requests and responses easier to manage, we'll declare a few classes to use for serializing / deserializing the JSON.

Go ahead and put their declarations before `Main`.

```
final class Transcription {
    public String name;
    public String description;
    public String locale;
    public URL recordingsUrl;
    public Hashtable<String, String> resultsUrls;
    public UUID id;
    public Date createdDateTime;
    public Date lastActionDateTime;
    public String status;
    public String statusMessage;
```

```

}

final class TranscriptionDefinition {
    private TranscriptionDefinition(String name, String description, String locale, URL recordingsUrl,
        ModelIdentity[] models) {
        this.Name = name;
        this.Description = description;
        this.RecordingsUrl = recordingsUrl;
        this.Locale = locale;
        this.Models = models;
        this.properties = new Hashtable<String, String>();
        this.properties.put("PunctuationMode", "DictatedAndAutomatic");
        this.properties.put("ProfanityFilterMode", "Masked");
        this.properties.put("AddWordLevelTimestamps", "True");
    }

    public String Name;
    public String Description;
    public URL RecordingsUrl;
    public String Locale;
    public ModelIdentity[] Models;
    public Dictionary<String, String> properties;

    public static TranscriptionDefinition Create(String name, String description, String locale, URL
recordingsUrl) {
        return TranscriptionDefinition.Create(name, description, locale, recordingsUrl, new
ModelIdentity[0]);
    }

    public static TranscriptionDefinition Create(String name, String description, String locale, URL
recordingsUrl,
        ModelIdentity[] models) {
        return new TranscriptionDefinition(name, description, locale, recordingsUrl, models);
    }
}

final class ModelIdentity {
    private ModelIdentity(UUID id) {
        this.Id = id;
    }

    public UUID Id;

    public static ModelIdentity Create(UUID Id) {
        return new ModelIdentity(Id);
    }
}

class AudioFileResult {
    public String AudioFileName;
    public SegmentResult[] SegmentResults;
}

class RootObject {
    public AudioFileResult[] AudioFileResults;
}

class NBest {
    public double Confidence;
    public String Lexical;
    public String ITN;
    public String MaskedITN;
    public String Display;
}

class SegmentResult {
    public String RecognitionStatus;
    public String Offset;
    public String Duration;
}

```

```
    public NBest[] NBest;  
}
```

Create and configure an Http Client

The first thing we'll need is an Http Client that has a correct base URL and authentication set. Insert this code in

Main

```
String url = "https://" + region + ".cris.ai/api/speechtotext/v2.0/Transcriptions/";  
URL serviceUrl = new URL(url);  
  
HttpURLConnection postConnection = (HttpURLConnection) serviceUrl.openConnection();  
postConnection.setDoOutput(true);  
postConnection.setRequestMethod("POST");  
postConnection.setRequestProperty("Content-Type", "application/json");  
postConnection.setRequestProperty("Ocp-Apim-Subscription-Key", subscriptionKey);
```

Generate a transcription request

Next, we'll generate the transcription request. Add this code to

```
TranscriptionDefinition definition = TranscriptionDefinition.Create(Name, Description, Locale,  
new URL(RecordingsBlobUri));
```

Send the request and check its status

Now we post the request to the Speech service and check the initial response code. This response code will simply indicate if the service has received the request. The service will return a Url in the response headers that's the location where it will store the transcription status.

```
Gson gson = new Gson();  
  
OutputStream stream = postConnection.getOutputStream();  
stream.write(gson.toJson(definition).getBytes());  
stream.flush();  
  
int statusCode = postConnection.getResponseCode();  
  
if (statusCode != HttpURLConnection.HTTP_ACCEPTED) {  
    System.out.println("Unexpected status code " + statusCode);  
    return;  
}
```

Wait for the transcription to complete

Since the service processes the transcription asynchronously, we need to poll for its status every so often. We'll check every 5 seconds.

We can check the status by retrieving the content at the Url we got when the posted the request. When we get the content back, we deserialize it into one of our helper class to make it easier to interact with.

Here's the polling code with status display for everything except a successful completion, we'll do that next.

```

String transcriptionLocation = postConnection.getHeaderField("location");

System.out.println("Transcription is located at " + transcriptionLocation);

URL transcriptionUrl = new URL(transcriptionLocation);

boolean completed = false;
while (!completed) {
    HttpURLConnection getConnection = (HttpURLConnection) transcriptionUrl.openConnection();
    getConnection.setRequestProperty("Ocp-Apim-Subscription-Key", subscriptionKey);
    getConnection.setRequestMethod("GET");

    int responseCode = getConnection.getResponseCode();

    if (responseCode != HttpURLConnection.HTTP_OK) {
        System.out.println("Fetching the transcription returned unexpected http code " + responseCode);
        return;
    }

    Transcription t = gson.fromJson(new InputStreamReader(getConnection.getInputStream()),
        Transcription.class);

    switch (t.status) {
        case "Failed":
            completed = true;
            System.out.println("Transcription has failed " + t.statusMessage);
            break;
        case "Succeeded":
            break;
        case "Running":
            System.out.println("Transcription is running.");
            break;
        case "NotStarted":
            System.out.println("Transcription has not started.");
            break;
    }

    if (!completed) {
        Thread.sleep(5000);
    }
}

```

Display the transcription results

Once the service has successfully completed the transcription the results will be stored in another Url that we can get from the status response.

We'll download the contents of that URL, deserialize the JSON, and loop through the results printing out the display text as we go.

```

package quickstart;

import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Date;
import java.util.Dictionary;
import java.util.Hashtable;
import java.util.UUID;

import com.google.gson.Gson;

```

```

final class Transcription {
    public String name;
    public String description;
    public String locale;
    public URL recordingsUrl;
    public Hashtable<String, String> resultsUrls;
    public UUID id;
    public Date createdDateTime;
    public Date lastActionDateTime;
    public String status;
    public String statusMessage;
}

final class TranscriptionDefinition {
    private TranscriptionDefinition(String name, String description, String locale, URL recordingsUrl,
        ModelIdentity[] models) {
        this.Name = name;
        this.Description = description;
        this.RecordingsUrl = recordingsUrl;
        this.Locale = locale;
        this.Models = models;
        this.properties = new Hashtable<String, String>();
        this.properties.put("PunctuationMode", "DictatedAndAutomatic");
        this.properties.put("ProfanityFilterMode", "Masked");
        this.properties.put("AddWordLevelTimestamps", "True");
    }

    public String Name;
    public String Description;
    public URL RecordingsUrl;
    public String Locale;
    public ModelIdentity[] Models;
    public Dictionary<String, String> properties;

    public static TranscriptionDefinition Create(String name, String description, String locale, URL
recordingsUrl) {
        return TranscriptionDefinition.Create(name, description, locale, recordingsUrl, new
ModelIdentity[0]);
    }

    public static TranscriptionDefinition Create(String name, String description, String locale, URL
recordingsUrl,
        ModelIdentity[] models) {
        return new TranscriptionDefinition(name, description, locale, recordingsUrl, models);
    }
}

final class ModelIdentity {
    private ModelIdentity(UUID id) {
        this.Id = id;
    }

    public UUID Id;

    public static ModelIdentity Create(UUID Id) {
        return new ModelIdentity(Id);
    }
}

class AudioFileResult {
    public String AudioFileName;
    public SegmentResult[] SegmentResults;
}

class RootObject {
    public AudioFileResult[] AudioFileResults;
}

```

```

class NBest {
    public double Confidence;
    public String Lexical;
    public String ITN;
    public String MaskedITN;
    public String Display;
}

class SegmentResult {
    public String RecognitionStatus;
    public String Offset;
    public String Duration;
    public NBest[] NBest;
}

public class Main {

    private static String region = "YourServiceRegion";
    private static String subscriptionKey = "YourSubscriptionKey";
    private static String Locale = "en-US";
    private static String RecordingsBlobUri = "YourFileUrl";
    private static String Name = "Simple transcription";
    private static String Description = "Simple transcription description";

    public static void main(String[] args) throws IOException, InterruptedException {
        System.out.println("Starting transcriptions client...");
        String url = "https://" + region + ".cris.ai/api/speechtotext/v2.0/Transcriptions/";
        URL serviceUrl = new URL(url);

        HttpURLConnection postConnection = (HttpURLConnection) serviceUrl.openConnection();
        postConnection.setDoOutput(true);
        postConnection.setRequestMethod("POST");
        postConnection.setRequestProperty("Content-Type", "application/json");
        postConnection.setRequestProperty("Ocp-Apim-Subscription-Key", subscriptionKey);

        TranscriptionDefinition definition = TranscriptionDefinition.Create(Name, Description, Locale,
            new URL(RecordingsBlobUri));

        Gson gson = new Gson();

        OutputStream stream = postConnection.getOutputStream();
        stream.write(gson.toJson(definition).getBytes());
        stream.flush();

        int statusCode = postConnection.getResponseCode();

        if (statusCode != HttpURLConnection.HTTP_ACCEPTED) {
            System.out.println("Unexpected status code " + statusCode);
            return;
        }

        String transcriptionLocation = postConnection.getHeaderField("location");

        System.out.println("Transcription is located at " + transcriptionLocation);

        URL transcriptionUrl = new URL(transcriptionLocation);

        boolean completed = false;
        while (!completed) {
            HttpURLConnection getConnection = (HttpURLConnection) transcriptionUrl.openConnection();
            getConnection.setRequestProperty("Ocp-Apim-Subscription-Key", subscriptionKey);
            getConnection.setRequestMethod("GET");

            int responseCode = getConnection.getResponseCode();

            if (responseCode != HttpURLConnection.HTTP_OK) {
                System.out.println("Fetching the transcription returned unexpected http code " +
responseCode);
            }
        }
    }
}

```

Check your code

At this point, your code should look like this: (We've added some comments to this version) [!code-java[]
(~/samples-cognitive-services-speech-sdk/quickstart/java/jre/from-blob/src/quickstart/Main.java)]

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

Next steps

[Explore Java samples on GitHub](#)

In this quickstart, you will use a REST API to recognize speech from files in a batch process. A batch process executes the speech transcription without any user interactions. It gives you a simple programming model, without the need to manage concurrency, custom speech models, or other details. It entails advanced control options, while making efficient use of Azure speech service resources.

The [batch transcription overview](#) describes the details to use this feature. The detailed API is available as a [Swagger document](#), under the heading `Custom Speech transcripts`.

The following quickstart will walk you through a usage sample.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Upload a source file to an azure blob](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Download and install the API client library

To execute the sample you need to generate the Python library for the REST API which is generated through [Swagger](#).

Follow these steps for the installation:

1. Go to <https://editor.swagger.io>.
2. Click **File**, then click **Import URL**.
3. Enter the Swagger URL including the region for your Speech service subscription:
`https://<your-region>.cris.ai/docs/v2.0/swagger`.
4. Click **Generate Client** and select **Python**.
5. Save the client library.
6. Extract the downloaded python-client-generated.zip somewhere in your file system.
7. Install the extracted python-client module in your Python environment using pip:
`pip install path/to/package/python-client`.
8. The installed package has the name `swagger_client`. You can check that the installation worked using the command `python -c "import swagger_client"`.

Note: Due to a [known bug in the Swagger autogeneration](#), you might encounter errors on importing the `swagger_client` package. These can be fixed by deleting the line with the content

```
from swagger_client.models.model import Model # noqa: F401,E501
```

from the file `swagger_client/models/model.py` and the line with the content

```
from swagger_client.models.inner_error import InnerError # noqa: F401,E501
```

from the file `swagger_client/models/inner_error.py` inside the installed package. The error message will tell you where these files are located for your installation.

Install other dependencies

The sample uses the `requests` library. You can install it with the command

```
pip install requests
```

Start with some boilerplate code

Let's add some code that works as a skeleton for our project.

```
#!/usr/bin/env python
# coding: utf-8
from typing import List

import logging
import sys
import requests
import time
import swagger_client as cris_client

logging.basicConfig(stream=sys.stdout, level=logging.DEBUG, format"%(message)s")

# Your subscription key and region for the speech service
SUBSCRIPTION_KEY = "YourSubscriptionKey"
SERVICE_REGION = "YourServiceRegion"

NAME = "Simple transcription"
DESCRIPTION = "Simple transcription description"

LOCALE = "en-US"
RECORDINGS_BLOB_URI = "<Your SAS Uri to the recording>"

# Set subscription information when doing transcription with custom models
ADAPTED_ACOUSTIC_ID = None # guid of a custom acoustic model
ADAPTED_LANGUAGE_ID = None # guid of a custom language model

def transcribe():
    logging.info("Starting transcription client...")

if __name__ == "__main__":
    transcribe()
```

(You'll need to replace the values of `YourSubscriptionKey`, `YourServiceRegion`, and `YourFileUrl` with your own values.)

Create and configure an Http Client

The first thing we'll need is an Http Client that has a correct base URL and authentication set. Insert this code in

transcribe

```
configuration = cris_client.Configuration()
configuration.api_key['Ocp-Apim-Subscription-Key'] = SUBSCRIPTION_KEY
configuration.host = "https://{}.cris.ai".format(SERVICE_REGION)

# create the client object and authenticate
client = cris_client.ApiClient(configuration)

# create an instance of the transcription api class
transcription_api = cris_client.CustomSpeechTranscriptionsApi(api_client=client)
```

Generate a transcription request

Next, we'll generate the transcription request. Add this code to

```
transcription_definition = cris_client.TranscriptionDefinition(
    name=NAME, description=DESCRIPTION, locale=LOCALE, recordings_url=RECORDINGS_BLOB_URI
)
```

Send the request and check its status

Now we post the request to the Speech service and check the initial response code. This response code will simply indicate if the service has received the request. The service will return a Url in the response headers that's the location where it will store the transcription status.

```
data, status, headers = transcription_api.create_transcription_with_http_info(transcription_definition)

# extract transcription location from the headers
transcription_location: str = headers["location"]

# get the transcription Id from the location URI
created_transcription: str = transcription_location.split('/')[-1]

logging.info("Created new transcription with id {}".format(created_transcription))
```

Wait for the transcription to complete

Since the service processes the transcription asynchronously, we need to poll for its status every so often. We'll check every 5 seconds.

We'll enumerate all the transcriptions that this Speech service resource is processing and look for the one we created.

Here's the polling code with status display for everything except a successful completion, we'll do that next.

```

logging.info("Checking status.")

completed = False

while not completed:
    running, not_started = 0, 0

    # get all transcriptions for the user
    transcriptions: List[cris_client.Transcription] = transcription_api.get_transcriptions()

    # for each transcription in the list we check the status
    for transcription in transcriptions:
        if transcription.status in ("Failed", "Succeeded"):
            # we check to see if it was the transcription we created from this client
            if created_transcription != transcription.id:
                continue

        completed = True

        if transcription.status == "Succeeded":
            else:
                logging.info("Transcription failed :{}.".format(transcription.status_message))
                break
        elif transcription.status == "Running":
            running += 1
        elif transcription.status == "NotStarted":
            not_started += 1

    logging.info("Transcriptions status: "
                "completed (this transcription): {}, {} running, {} not started yet".format(
                    completed, running, not_started))

# wait for 5 seconds
time.sleep(5)

```

Display the transcription results

Once the service has successfully completed the transcription the results will be stored in another Url that we can get from the status response.

Here we get that result JSON and display it.

```

results_uri = transcription.results_urls["channel_0"]
results = requests.get(results_uri)
logging.info("Transcription succeeded. Results: ")
logging.info(results.content.decode("utf-8"))

```

Check your code

At this point, your code should look like this: (We've added some comments to this version)

```

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

# Copyright (c) Microsoft. All rights reserved.
# Licensed under the MIT license. See LICENSE.md file in the project root for full license information.

from typing import List

import logging
import sys
import requests

```

```

import time
import swagger_client as cris_client


logging.basicConfig(stream=sys.stdout, level=logging.DEBUG, format"%(message)s")

# Your subscription key and region for the speech service
SUBSCRIPTION_KEY = "YourSubscriptionKey"
SERVICE_REGION = "YourServiceRegion"

NAME = "Simple transcription"
DESCRIPTION = "Simple transcription description"

LOCALE = "en-US"
RECORDINGS_BLOB_URI = "<Your SAS Uri to the recording>"

# Set subscription information when doing transcription with custom models
ADAPTED_ACOUSTIC_ID = None # guid of a custom acoustic model
ADAPTED_LANGUAGE_ID = None # guid of a custom language model

def transcribe():
    logging.info("Starting transcription client...")

    # configure API key authorization: subscription_key
    configuration = cris_client.Configuration()
    configuration.api_key['Ocp-Apim-Subscription-Key'] = SUBSCRIPTION_KEY
    configuration.host = "https://{}.cris.ai".format(SERVICE_REGION)

    # create the client object and authenticate
    client = cris_client.ApiClient(configuration)

    # create an instance of the transcription api class
    transcription_api = cris_client.CustomSpeechTranscriptionsApi(api_client=client)

    # Use base models for transcription. Comment this block if you are using a custom model.
    # Note: you can specify additional transcription properties by passing a
    # dictionary in the properties parameter. See
    # https://docs.microsoft.com/azure/cognitive-services/speech-service/batch-transcription
    # for supported parameters.
    transcription_definition = cris_client.TranscriptionDefinition(
        name=NAME, description=DESCRIPTION, locale=LOCALE, recordings_url=RECORDINGS_BLOB_URI
    )

    # Uncomment this block to use custom models for transcription.
    # Model information (ADAPTED_ACOUSTIC_ID and ADAPTED_LANGUAGE_ID) must be set above.
    # if ADAPTED_ACOUSTIC_ID is None or ADAPTED_LANGUAGE_ID is None:
    #     logging.info("Custom model ids must be set to when using custom models")
    #     transcription_definition = cris_client.TranscriptionDefinition(
    #         name=NAME, description=DESCRIPTION, locale=LOCALE, recordings_url=RECORDINGS_BLOB_URI,
    #         models=[cris_client.ModelIdentity(ADAPTED_ACOUSTIC_ID),
    #                cris_client.ModelIdentity(ADAPTED_LANGUAGE_ID)]
    #     )

    data, status, headers = transcription_api.create_transcription_with_http_info(transcription_definition)

    # extract transcription location from the headers
    transcription_location: str = headers["location"]

    # get the transcription Id from the location URI
    created_transcription: str = transcription_location.split('/')[-1]

    logging.info("Created new transcription with id {}".format(created_transcription))

    logging.info("Checking status.")

    completed = False

    while not completed:

```

```

while not completed:
    running, not_started = 0, 0

    # get all transcriptions for the user
    transcriptions: List[cris_client.Transcription] = transcription_api.get_transcriptions()

    # for each transcription in the list we check the status
    for transcription in transcriptions:
        if transcription.status in ("Failed", "Succeeded"):
            # we check to see if it was the transcription we created from this client
            if created_transcription != transcription.id:
                continue

        completed = True

        if transcription.status == "Succeeded":
            results_uri = transcription.results_urls["channel_0"]
            results = requests.get(results_uri)
            logging.info("Transcription succeeded. Results: ")
            logging.info(results.content.decode("utf-8"))
        else:
            logging.info("Transcription failed :{}".format(transcription.status_message))
            break
        elif transcription.status == "Running":
            running += 1
        elif transcription.status == "NotStarted":
            not_started += 1

    logging.info("Transcriptions status: "
                "completed (this transcription): {}, {} running, {} not started yet".format(
                    completed, running, not_started))

    # wait for 5 seconds
    time.sleep(5)

    input("Press any key...")

if __name__ == "__main__":
    transcribe()

```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

Next steps

[Explore Python samples on GitHub](#)

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	ADDITIONAL QUICKSTARTS	CODE SAMPLES
C++		Quickstarts , Samples

LANGUAGE	ADDITIONAL QUICKSTARTS	CODE SAMPLES
C#		.NET Framework, .NET Core, UWP, Unity, Xamarin
Java		Android, JRE
Javascript		Browser
Node.js		Windows, Linux, macOS
Objective-C	macOS, iOS	iOS, macOS
Python		Windows, Linux, macOS
Swift	macOS, iOS	iOS, macOS

Specify source language for speech to text

11/14/2019 • 2 minutes to read • [Edit Online](#)

In this article, you'll learn how to specify the source language for an audio input passed to the Speech SDK for speech recognition. Additionally, example code is provided to specify a custom speech model for improved recognition.

How to specify source language in C#

The first step is to create a `SpeechConfig` :

```
var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

Next, specify the source language of your audio with `SpeechRecognitionLanguage` :

```
speechConfig.SpeechRecognitionLanguage = "de-DE";
```

If you're using a custom model for recognition, you can specify the endpoint with `EndpointId` :

```
speechConfig.EndpointId = "The Endpoint ID for your custom model.;"
```

How to specify source language in C++

In this example, the source language is provided explicitly as a parameter using the `FromConfig` method.

```
auto recognizer = SpeechRecognizer::FromConfig(speechConfig, "de-DE", audioConfig);
```

In this example, the source language is provided using `SourceLanguageConfig`. Then, the `sourceLanguageConfig` is passed as a parameter to `FromConfig` when creating the `recognizer`.

```
auto sourceLanguageConfig = SourceLanguageConfig::FromLanguage("de-DE");
auto recognizer = SpeechRecognizer::FromConfig(speechConfig, sourceLanguageConfig, audioConfig);
```

In this example, the source language and custom endpoint are provided using `SourceLanguageConfig`. The `sourceLanguageConfig` is passed as a parameter to `FromConfig` when creating the `recognizer`.

```
auto sourceLanguageConfig = SourceLanguageConfig::FromLanguage("de-DE", "The Endpoint ID for your custom
model.");
auto recognizer = SpeechRecognizer::FromConfig(speechConfig, sourceLanguageConfig, audioConfig);
```

NOTE

`SetSpeechRecognitionLanguage` and `SetEndpointId` are deprecated methods from the `SpeechConfig` class in C++ and Java. The use of these methods are discouraged, and shouldn't be used when constructing a `SpeechRecognizer`.

How to specify source language in Java

In this example, the source language is provided explicitly when creating a new `SpeechRecognizer`.

```
SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, "de-DE", audioConfig);
```

In this example, the source language is provided using `SourceLanguageConfig`. Then, the `sourceLanguageConfig` is passed as a parameter when creating a new `SpeechRecognizer`.

```
SourceLanguageConfig sourceLanguageConfig = SourceLanguageConfig.fromLanguage("de-DE");
SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, sourceLanguageConfig, audioConfig);
```

In this example, the source language and custom endpoint are provided using `SourceLanguageConfig`. Then, the `sourceLanguageConfig` is passed as a parameter when creating a new `SpeechRecognizer`.

```
SourceLanguageConfig sourceLanguageConfig = SourceLanguageConfig.fromLanguage("de-DE", "The Endpoint ID for
your custom model.");
SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, sourceLanguageConfig, audioConfig);
```

NOTE

`setSpeechRecognitionLanguage` and `setEndpointId` are deprecated methods from the `SpeechConfig` class in C++ and Java. The use of these methods are discouraged, and shouldn't be used when constructing a `SpeechRecognizer`.

How to specify source language in Python

The first step is to create a `speech_config`:

```
speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"
speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)
```

Next, specify the source language of your audio with `speech_recognition_language`:

```
speech_config.speech_recognition_language="de-DE"
```

If you're using a custom model for recognition, you can specify the endpoint with `endpoint_id`:

```
speech_config.endpoint_id = "The Endpoint ID for your custom model."
```

How to specify source language in Javascript

The first step is to create a `SpeechConfig`:

```
var speechConfig = sdk.SpeechConfig.fromSubscription("YourSubscriptionkey", "YourRegion");
```

Next, specify the source language of your audio with `speechRecognitionLanguage`:

```
speechConfig.speechRecognitionLanguage = "de-DE";
```

If you're using a custom model for recognition, you can specify the endpoint with `endpointId` :

```
speechConfig.endpointId = "The Endpoint ID for your custom model.;"
```

How to specify source language in Objective-C

The first step is to create a `speechConfig` :

```
SPXSpeechConfiguration *speechConfig = [[SPXSpeechConfiguration alloc]  
initWithSubscription:@"YourSubscriptionkey" region:@"YourRegion"];
```

Next, specify the source language of your audio with `speechRecognitionLanguage` :

```
speechConfig.speechRecognitionLanguage = @"de-DE";
```

If you're using a custom model for recognition, you can specify the endpoint with `endpointId` :

```
speechConfig.endpointId = @"The Endpoint ID for your custom model.;"
```

See also

- For a list of supported languages and locales for speech to text, see [Language support](#).

Next steps

- [Speech SDK reference documentation](#)

What is Custom Speech?

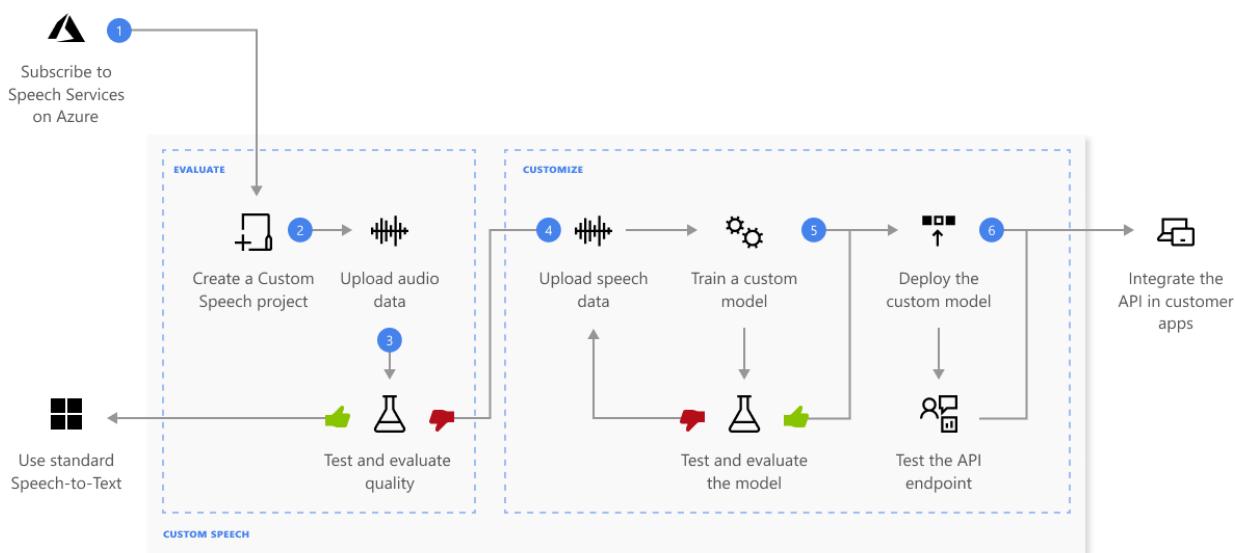
12/4/2019 • 3 minutes to read • [Edit Online](#)

Custom Speech is a set of online tools that allow you to evaluate and improve Microsoft's speech-to-text accuracy for your applications, tools, and products. All it takes to get started are a handful of test audio files. Follow the links below to start creating a custom speech-to-text experience.

What's in Custom Speech?

Before you can do anything with Custom Speech, you'll need an Azure account and a Speech service subscription. Once you've got an account, you can prep your data, train and test your models, inspect recognition quality, evaluate accuracy, and ultimately deploy and use the custom speech-to-text model.

This diagram highlights the pieces that make up the [Custom Speech portal](#). Use the links below to learn more about each step.



1. **Subscribe and create a project** - Create an Azure account and subscribe to the Speech service. This unified subscription gives you access to speech-to-text, text-to-speech, speech translation, and the [Custom Speech portal](#). Then, using your Speech service subscription, create your first Custom Speech project.
2. **Upload test data** - Upload test data (audio files) to evaluate Microsoft's speech-to-text offering for your applications, tools, and products.
3. **Inspect recognition quality** - Use the [Custom Speech portal](#) to play back uploaded audio and inspect the speech recognition quality of your test data. For quantitative measurements, see [Inspect data](#).
4. **Evaluate accuracy** - Evaluate the accuracy of the speech-to-text model. The [Custom Speech portal](#) will provide a *Word Error Rate*, which can be used to determine if additional training is required. If you're satisfied with the accuracy, you can use the Speech service APIs directly. If you'd like to improve accuracy by a relative average of 5% - 20%, use the **Training** tab in the portal to upload additional training data, such as human-labeled transcripts and related text.
5. **Train the model** - Improve the accuracy of your speech-to-text model by providing written transcripts (10-1,000 hours) and related text (<200 MB) along with your audio test data. This data helps to train the speech-to-text model. After training, retest, and if you're satisfied with the result, you can deploy your model.

6. [Deploy the model](#) - Create a custom endpoint for your speech-to-text model and use it in your applications, tools, or products.

Set up your Azure account

A Speech service subscription is required before you can use the [Custom Speech portal](#) to create a custom model. Follow these instructions to create a standard Speech service subscription: [Create a Speech Subscription](#).

NOTE

Please be sure to create standard (S0) subscriptions, free trial (F0) subscriptions are not supported.

Once you've created an Azure account and a Speech service subscription, you'll need to sign in to [Custom Speech portal](#) and connect your subscription.

1. Get your Speech service subscription key from the Azure portal.
2. Sign-in to the [Custom Speech portal](#).
3. Select the subscription you need to work on and create a speech project.
4. If you'd like to modify your subscription, use the  icon located in the top navigation.

How to create a project

Content like data, models, tests, and endpoints are organized into **Projects** in the [Custom Speech portal](#). Each project is specific to a domain and country/language. For example, you may create a project for call centers that use English in the United States.

To create your first project, select the **Speech-to-text/Custom speech**, then click **New Project**. Follow the instructions provided by the wizard to create your project. After you've created a project, you should see four tabs: **Data, Testing, Training, and Deployment**. Use the links provided in [Next steps](#) to learn how to use each tab.

Next steps

- [Prepare and test your data](#)
- [Inspect your data](#)
- [Evaluate your data](#)
- [Train your model](#)
- [Deploy your model](#)

Phrase Lists for speech-to-text

12/9/2019 • 2 minutes to read • [Edit Online](#)

By providing the Speech service with a list of phrases, you can improve the accuracy of speech recognition. Phrase Lists are used to identify known phrases in audio data, like a person's name or a specific location.

As an example, if you have a command "Move to" and a possible destination of "Ward" that may be spoken, you can add an entry of "Move to Ward". Adding a phrase will increase the probability that when the audio is recognized that "Move to Ward" will be recognized instead of "Move toward".

Single words or complete phrases can be added to a Phrase List. During recognition, an entry in a phrase list is used if an exact match for the entire phrase is included in the audio as a separate phrase. If an exact match to the phrase is not found, recognition is not assisted.

NOTE

Currently, Phrase Lists supports only English for speech-to-text.

How to use Phrase Lists

The samples below illustrate how to build a Phrase List using the `PhraseListGrammar` object.

```
PhraseListGrammar phraseList = PhraseListGrammar.FromRecognizer(recognizer);
phraseList.AddPhrase("Move to Ward");
phraseList.AddPhrase("Move to Bill");
phraseList.AddPhrase("Move to Ted");
```

```
auto phraselist = PhraseListGrammar::FromRecognizer(recognizer);
phraselist->AddPhrase("Move to Ward");
phraselist->AddPhrase("Move to Bill");
phraselist->AddPhrase("Move to Ted");
```

```
PhraseListGrammar phraseListGrammar = PhraseListGrammar.fromRecognizer(recognizer);
phraseListGrammar.addPhrase("Move to Ward");
phraseListGrammar.addPhrase("Move to Bill");
phraseListGrammar.addPhrase("Move to Ted");
```

```
phrase_list_grammar = speechsdk.PhraseListGrammar.from_recognizer(reco)
phrase_list_grammar.addPhrase("Move to Ward")
phrase_list_grammar.addPhrase("Move to Bill")
phrase_list_grammar.addPhrase("Move to Ted")
```

```
var phraseListGrammar = SpeechSDK.PhraseListGrammar.fromRecognizer(reco);
phraseListGrammar.addPhrase("Move to Ward");
phraseListGrammar.addPhrase("Move to Bill");
phraseListGrammar.addPhrase("Move to Ted");
```

NOTE

The maximum number of Phrase Lists that the Speech service will use to match speech is 1024 phrases.

You can also clear the phrases associated with the `PhraseListGrammar` by calling `clear()`.

```
phraseList.Clear();
```

```
phraselist->Clear();
```

```
phraseListGrammar.clear();
```

```
phrase_list_grammar.clear()
```

```
phraseListGrammar.clear();
```

NOTE

Changes to a `PhraseListGrammar` object take effect on the next recognition or following a reconnection to the Speech service.

Next steps

- [Speech SDK reference documentation](#)

Tutorial: Create a tenant model (preview)

12/13/2019 • 6 minutes to read • [Edit Online](#)

Tenant Model (Custom Speech with Office 365 data) is an opt-in service for Office 365 enterprise customers that automatically generates a custom speech recognition model from your organization's Office 365 data. The model is optimized for technical terms, jargon, and people's names, all in a secure and compliant way.

IMPORTANT

If your organization enrolls by using the Tenant Model service, Speech Service may access your organization's language model. The model is generated from Office 365 public group emails and documents, which can be seen by anyone in your organization. Your organization's Office 365 admin can turn on or turn off the use of the organization-wide language model from the Office 365 admin portal.

In this tutorial, you'll learn how to:

- Enroll in the Tenant Model by using the Microsoft 365 admin center
- Get a Speech subscription key
- Create a tenant model
- Deploy a tenant model
- Use your tenant model with the Speech SDK

Enroll in the Tenant Model service

Before you can deploy your tenant model, you need to be enrolled in the Tenant Model service. Enrollment is completed in the Microsoft 365 admin center and can be done only by your Microsoft 365 admin.

1. Sign in to the [Microsoft 365 admin center](#).
2. In the left pane, select **Settings**, select **Apps**, and then select **Azure Speech Services**.

The screenshot shows the Microsoft 365 admin center interface. On the left, there is a navigation sidebar with options like Home, Users, Groups, Roles, Billing, Support, and Settings. The 'Settings' and 'Apps' items under 'Settings' are highlighted with red boxes. The main content area is titled 'Services & add-ins' and lists various services. The 'Azure Speech Services' item is also highlighted with a red box. A table below lists the service details:

Name ↑	Description	Host Apps
Azure multi-factor authentication	Manage multi-factor authentication settings for your users.	
Azure Speech Services	Allow use of your organization's emails and documents to improve speech recognition accuracy	
Cortana	Turn Cortana access on or off for your entire organization.	
Directory Synchronization	Sync users to the cloud using Azure Active Directory.	
Dynamics 365 AI for Sales - Analytics	Allow Dynamics 365 to generate insights based on user data.	
Dynamics 365 AI for Sales - Connection Graph	Manage and update your Dynamics 365 AI for Sales - Connection Graph settings	
Integrated apps	Let users decide whether third-party apps can access their Office 365 info.	
Microsoft Graph data connect	Manage and update your Microsoft Graph data connect settings	

3. Select the **Allow the organization-wide language model** check box, and then select **Save changes**.

The screenshot shows the Microsoft 365 admin center interface. On the left, there's a navigation sidebar with various options like Home, Users, Groups, Roles, Billing, Support, Settings, Microsoft Search, Apps, Security & privacy, and Organization profile. The main content area is titled 'Services & add-ins' and lists several services: Azure multi-factor authentication, Azure Speech Services, Cortana, Directory Synchronization, Dynamics 365 AI for Sales - Analytics, Dynamics 365 AI for Sales - Connection Graph, and Integrated apps. The 'Azure Speech Services' row has a detailed description: 'Speech recognition performs better if it understands terminology, phrases, or acronyms commonly used within your organization. These are referred to as language models. This setting allows Azure Speech Services to access organization-wide language models based on emails and documents shared within your organization. This language model improves speech recognition accuracy for O365 and non-Microsoft applications enrolled in Azure Speech Services (called Organization Speech Model). The organization-wide language model is built only from 1) Emails sent to public groups that everyone in your organization can join, 2) Documents saved to the Microsoft SharePoint sites of these public groups that everyone in your organization can access.' Below this description are two buttons: 'Allow the organization-wide language model' (with a checked checkbox) and 'Save changes' (highlighted with a red box).

To turn off the tenant model instance:

1. Repeat the preceding steps 1 and 2.
2. Clear the **Allow the organization-wide language model** check box, and then select **Save changes**.

Get a Speech subscription key

To use your tenant model with the Speech SDK, you need a Speech resource and its associated subscription key.

1. Sign in to the [Azure portal](#).
2. Select **Create a resource**.
3. In the **Search** box, type **Speech**.
4. In the results list, select **Speech**, and then select **Create**.
5. Follow the onscreen instructions to create your resource. Make sure that:
 - **Location** is set to either **eastus** or **westus**.
 - **Pricing tier** is set to **S0**.
6. Select **Create**.

After a few minutes, your resource is created. The subscription key is available in the **Overview** section for your resource.

Create a language model

After your admin has enabled Tenant Model for your organization, you can create a language model that's based on your Office 365 data.

1. Sign in to [Speech Studio](#).
2. At the top right, select **Settings** (gear icon), and then select **Tenant Model settings**.

The screenshot shows the Cognitive Services | Speech Customization page in Speech Studio. At the top, there's a header with icons for search, settings, help, and user profile. Below the header, there are two sections: 'Subscription' and 'Tenant Model settings'. The 'Tenant Model settings' link is highlighted with a red box.

Speech Studio displays a message that lets you know whether you're qualified to create a tenant model.

NOTE

Office 365 enterprise customers in North America are eligible to create a tenant model (English). If you're a Customer Lockbox, Customer Key, or Office 365 Government customer, this feature isn't available. To determine whether you're a Customer Lockbox or Customer Key customer, see:

- [Customer Lockbox](#)
- [Customer Key](#)
- [Office 365 Government](#)

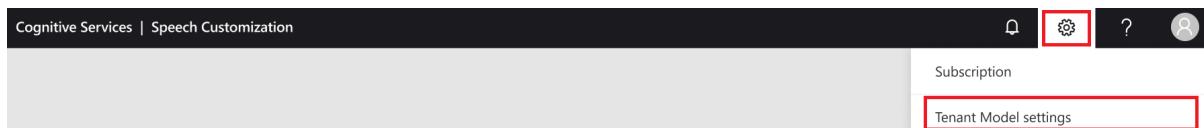
3. Select **Opt in**.

When your tenant model is ready, you'll receive a confirmation email message with further instructions.

Deploy your tenant model

When your tenant model instance is ready, deploy it by doing the following:

1. In your confirmation email message, select the **View model** button. Or sign in to [Speech Studio](#).
2. At the top right, select **Settings** (gear icon), and then select **Tenant Model settings**.



3. Select **Deploy**.

When your model has been deployed, the status changes to *Deployed*.

Use your tenant model with the Speech SDK

Now that you've deployed your model, you can use it with the Speech SDK. In this section, you use sample code to call Speech Service by using Azure Active Directory (Azure AD) authentication.

Let's look at the code you'll use to call the Speech SDK in C#. In this example, you perform speech recognition by using your tenant model. This guide presumes that your platform is already set up. If you need setup help, see [Quickstart: Recognize speech, C# \(.NET Core\)](#).

Copy this code into your project:

```
namespace PrincetonSROnly.FrontEnd.Samples
{
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Net.Http;
    using System.Text;
    using System.Text.RegularExpressions;
    using System.Threading.Tasks;
    using Microsoft.CognitiveServices.Speech;
    using Microsoft.CognitiveServices.Speech.Audio;
    using Microsoft.IdentityModel.Clients.ActiveDirectory;
    using Newtonsoft.Json.Linq;

    // ServiceApplicationId is a fixed value. No need to change it.

    public class TenantLMSample
    {
        private const string EndpointUriArgName = "EndpointUri";
```

```

private const string SubscriptionKeyArgName = "SubscriptionKey";
private const string UsernameArgName = "Username";
private const string PasswordArgName = "Password";
private const string ClientApplicationId = "f87bc118-1576-4097-93c9-dbf8f45ef0dd";
private const string ServiceApplicationId = "18301695-f99d-4cae-9618-6901d4bdc7be";

public static async Task ContinuousRecognitionWithTenantLMAsync(Uri endpointUri, string
subscriptionKey, string audioDirPath, string username, string password)
{
    var config = SpeechConfig.FromEndpoint(endpointUri, subscriptionKey);

    // Passing client specific information for obtaining LM
    if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
    {
        config.AuthorizationToken = await
AcquireAuthTokenWithInteractiveLoginAsync().ConfigureAwait(false);
    }
    else
    {
        config.AuthorizationToken = await AcquireAuthTokenWithUsernamePasswordAsync(username,
password).ConfigureAwait(false);
    }

    var stopRecognition = new TaskCompletionSource<int>();

    // Creates a speech recognizer using file as audio input.
    // Replace with your own audio file name.
    using (var audioInput = AudioConfig.FromWavFileInput(audioDirPath))
    {
        using (var recognizer = new SpeechRecognizer(config, audioInput))
        {
            // Subscribes to events
            recognizer.Recognizing += (s, e) =>
            {
                Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
            };

            recognizer.Recognized += (s, e) =>
            {
                if (e.Result.Reason == ResultReason.RecognizedSpeech)
                {
                    Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                }
                else if (e.Result.Reason == ResultReason.NoMatch)
                {
                    Console.WriteLine($"NOMATCH: Speech could not be recognized.");
                }
            };
        };

        recognizer.Canceled += (s, e) =>
        {
            Console.WriteLine($"CANCELED: Reason={e.Reason}");
            if (e.Reason == CancellationReason.Error)
            {
                Exception exp = new Exception(string.Format("Error Code: {0}\nError Details{1}\nIs
your subscription information updated?", e.ErrorCode, e.ErrorDetails));
                throw exp;
            }

            stopRecognition.TrySetResult(0);
        };

        recognizer.SessionStarted += (s, e) =>
        {
            Console.WriteLine("\n      Session started event.");
        };

        recognizer.SessionStopped += (s, e) =>
        {

```

```

        Console.WriteLine("\n      Session stopped event.");
        Console.WriteLine("\nStop recognition.");
        stopRecognition.TrySetResult(0);
    };

    // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop
recognition.
    await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

    // Waits for completion.
    // Use Task.WaitAny to keep the task rooted.
    Task.WaitAny(new[] { stopRecognition.Task });

    // Stops recognition.
    await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
}
}

public static void Main(string[] args)
{
    var arguments = new Dictionary<string, string>();
    string inputArgNamePattern = "--";
    Regex regex = new Regex(inputArgNamePattern);
    if (args.Length > 0)
    {
        foreach (var arg in args)
        {
            var userArgs = arg.Split("=");
            arguments[regex.Replace(userArgs[0], string.Empty)] = userArgs[1];
        }
    }

    var endpointString = arguments.GetValueOrDefault(EndpointUriArgName,
$"wss://westus.online.princeton.customspeech.ai/msgraphcustomspeech/conversation/v1");
    var endpointUri = new Uri(endpointString);

    if (!arguments.ContainsKey(SubscriptionKeyArgName))
    {
        Exception exp = new Exception("Subscription Key missing! Please pass in a Cognitive services
subscription Key using --SubscriptionKey=\"your_subscription_key\" +
                    "Find more information on creating a Cognitive services resource and accessing your
Subscription key here: https://docs.microsoft.com/azure/cognitive-services/cognitive-services-apis-create-account?tabs=multiservice%2Cwindows";
        throw exp;
    }

    var subscriptionKey = arguments[SubscriptionKeyArgName];
    var username = arguments.GetValueOrDefault(UsernameArgName, null);
    var password = arguments.GetValueOrDefault>PasswordArgName, null);

    var audioDirPath =
Path.Combine(Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location),
"../../../../AudioSamples/DictationBatman.wav");
    if (!File.Exists(audioDirPath))
    {
        Exception exp = new Exception(string.Format("Audio File does not exist at path: {0}",
audioDirPath));
        throw exp;
    }

    ContinuousRecognitionWithTenantLMAsync(endpointUri, subscriptionKey, audioDirPath, username,
password).GetAwaiter().GetResult();
}

private static async Task<string> AcquireAuthTokenWithUsernamePasswordAsync(string username, string
password)
{
    var tokenEndpoint = "https://login.microsoftonline.com/common/oauth2/token";
}

```

```

        var postBody = $"resource={ServiceApplicationId}&client_id={ClientApplicationId}&grant_type=password&username={username}&password={password}";
        var stringContent = new StringContent(postBody, Encoding.UTF8, "application/x-www-form-urlencoded");
        using (HttpClient httpClient = new HttpClient())
        {
            var response = await httpClient.PostAsync(tokenEndpoint, stringContent).ConfigureAwait(false);

            if (response.IsSuccessStatusCode)
            {
                var result = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

                JObject jobject = JObject.Parse(result);
                return jobject["access_token"].Value<string>();
            }
            else
            {
                throw new Exception($"Requesting token from {tokenEndpoint} failed with status code {response.StatusCode}: {await response.Content.ReadAsStringAsync().ConfigureAwait(false)}");
            }
        }
    }

    private static async Task<string> AcquireAuthTokenWithInteractiveLoginAsync()
    {
        var authContext = new AuthenticationContext("https://login.windows.net/microsoft.onmicrosoft.com");
        var deviceCodeResult = await authContext.AcquireDeviceCodeAsync(ServiceApplicationId, ClientApplicationId).ConfigureAwait(false);

        Console.WriteLine(deviceCodeResult.Message);

        var authResult = await authContext.AcquireTokenByDeviceCodeAsync(deviceCodeResult).ConfigureAwait(false);

        return authResult.AccessToken;
    }
}
}

```

Next, you need to rebuild and run the project from the command line. Before you run the command, update a few parameters by doing the following:

1. Replace <Username> and <Password> with the values for a valid tenant user.
2. Replace <Subscription-Key> with the subscription key for your Speech resource. This value is available in the **Overview** section for your Speech resource in the [Azure portal](#).
3. Replace <Endpoint-Uri> with the following endpoint. Make sure that you replace <your region> with the region where your Speech resource was created. These regions are supported: <westus>, <westus2>, and <eastus>. Your region information is available in the **Overview** section of your Speech resource in the [Azure portal](#).

```
"wss://{your region}.online.princeton.customspeech.ai/msgraphcustomspeech/conversation/v1".
```

4. Run the following command:

```
dotnet TenantLMSample.dll --Username=<Username> --Password=<Password> --SubscriptionKey=<Subscription-Key> --EndpointUri=<Endpoint-Uri>
```

In this tutorial, you've learned how to use Office 365 data to create a custom speech recognition model, deploy it, and use it with the Speech SDK.

Next steps

- [Speech Studio](#)
- [Speech SDK](#)

How to: Select an audio input device with the Speech SDK

11/14/2019 • 5 minutes to read • [Edit Online](#)

Version 1.3.0 of the Speech SDK introduces an API to select the audio input. This article describes how to obtain the IDs of the audio devices connected to a system. These can then be used in the Speech SDK by configuring the audio device through the `AudioConfig` object:

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

```
audioConfig = AudioConfig.FromMicrophoneInput("<device id>");
```

```
audio_config = AudioConfig(device_name="<device id>");
```

```
audioConfig = AudioConfiguration.FromMicrophoneInput("<device id>");
```

```
audioConfig = AudioConfiguration.fromMicrophoneInput("<device id>");
```

```
audioConfig = AudioConfiguration.fromMicrophoneInput("<device id>");
```

NOTE

Microphone usage is not available for JavaScript running in Node.js

Audio device IDs on Windows for Desktop applications

Audio device [endpoint ID strings](#) can be retrieved from the `IMMDevice` object in Windows for Desktop applications.

The following code sample illustrates how to use it to enumerate audio devices in C++:

```
#include <cstdio>
#include <mmdeviceapi.h>

#include <FunctionDiscoveryKeys_devkey.h>

const CLSID CLSID_MMDeviceEnumerator = __uuidof(MMDeviceEnumerator);
const IID IID_IMMDeviceEnumerator = __uuidof(IMMDeviceEnumerator);

constexpr auto REFTIMES_PER_SEC = (10000000 * 25);
constexpr auto REFTIMES_PER_MILLISEC = 10000;

#define EXIT_ON_ERROR(hres) \
    if (FAILED(hres)) { goto Exit; }

#define SAFE_RELEASE(punk) \
    if ((punk) != NULL) \
        punk->Release();
```

```

    { (punk)->Release(); (punk) = NULL; }

void ListEndpoints();

int main()
{
    CoInitializeEx(NULL, COINIT_MULTITHREADED);
    ListEndpoints();
}

//-----
// This function enumerates all active (plugged in) audio
// rendering endpoint devices. It prints the friendly name
// and endpoint ID string of each endpoint device.
//-----
void ListEndpoints()
{
    HRESULT hr = S_OK;
    IMMDeviceEnumerator *pEnumerator = NULL;
    IMMDeviceCollection *pCollection = NULL;
    IMMDevice *pEndpoint = NULL;
    IPropertyStore *pProps = NULL;
    LPWSTR pwszID = NULL;

    hr = CoCreateInstance(CLSID_MMDeviceEnumerator, NULL, CLSCTX_ALL, IID_IMMDeviceEnumerator,
    (void**)&pEnumerator);
    EXIT_ON_ERROR(hr);

    hr = pEnumerator->EnumAudioEndpoints(eCapture, DEVICE_STATE_ACTIVE, &pCollection);
    EXIT_ON_ERROR(hr);

    UINT count;
    hr = pCollection->GetCount(&count);
    EXIT_ON_ERROR(hr);

    if (count == 0)
    {
        printf("No endpoints found.\n");
    }

    // Each iteration prints the name of an endpoint device.
    PROPVARIANT varName;
    for (ULONG i = 0; i < count; i++)
    {
        // Get pointer to endpoint number i.
        hr = pCollection->Item(i, &pEndpoint);
        EXIT_ON_ERROR(hr);

        // Get the endpoint ID string.
        hr = pEndpoint->GetId(&pwszID);
        EXIT_ON_ERROR(hr);

        hr = pEndpoint->OpenPropertyStore(
            STGM_READ, &pProps);
        EXIT_ON_ERROR(hr);

        // Initialize container for property value.
        PropVariantInit(&varName);

        // Get the endpoint's friendly-name property.
        hr = pProps->GetValue(PKEY_Device_FriendlyName, &varName);
        EXIT_ON_ERROR(hr);

        // Print endpoint friendly name and endpoint ID.
        printf("Endpoint %d: \"%S\" (%S)\n", i, varName.pwszVal, pwszID);
    }

Exit:
    CoTaskMemFree(pwszID);
}

```

```

pwszID = NULL;
PropVariantClear(&varName);
SAFE_RELEASE(pEnumerator);
SAFE_RELEASE(pCollection);
SAFE_RELEASE(pEndpoint);
SAFE_RELEASE(pProps);
}

```

In C#, the [NAudio](#) library can be used to access the CoreAudio API and enumerate devices as follows:

```

using System;

using NAudio.CoreAudioApi;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            var enumerator = new MMDeviceEnumerator();
            foreach (var endpoint in
                enumerator.EnumerateAudioEndPoints(DataFlow.Capture, DeviceState.Active))
            {
                Console.WriteLine("{0} ({1})", endpoint.FriendlyName, endpoint.ID);
            }
        }
    }
}

```

A sample device ID is `{0.0.1.00000000}.{5f23ab69-6181-4f4a-81a4-45414013aac8}`.

Audio device IDs on UWP

On the Universal Windows Platform (UWP), audio input devices can be obtained using the `Id()` property of the corresponding [DeviceInformation](#) object.

The following code samples show how to do this in C++ and C#:

```

#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Devices.Enumeration.h>

using namespace winrt::Windows::Devices::Enumeration;

void enumerateDeviceIds()
{
    auto promise = DeviceInformation::FindAllAsync(DeviceClass::AudioCapture);

    promise.Completed(
        []([winrt::Windows::Foundation::IAsyncOperation<DeviceInformationCollection> const& sender,
            winrt::Windows::Foundation::AsyncStatus /* asyncStatus */) {
            auto info = sender.GetResults();
            auto num_devices = info.Size();

            for (const auto &device : info)
            {
                std::wstringstream ss{};
                ss << "looking at device (of " << num_devices << "): " << device.Id().c_str() << "\n";
                OutputDebugString(ss.str().c_str());
            }
        });
}

```

```

using Windows.Devices.Enumeration;
using System.Linq;

namespace helloworld {
    private async void EnumerateDevices()
    {
        var devices = await DeviceInformation.FindAllAsync(DeviceClass.AudioCapture);

        foreach (var device in devices)
        {
            Console.WriteLine($"{device.Name}, {device.Id}\n");
        }
    }
}

```

A sample device ID is

```
\\\?\SWD#MMDEVAPI#{0.0.1.00000000}.{5f23ab69-6181-4f4a-81a4-45414013aac8}#{2eef81be-33fa-4800-9670-1cd474972c3f}
```

Audio device IDs on Linux

The device IDs are selected using standard ALSA device IDs.

The IDs of the inputs attached to the system are contained in the output of the command `arecord -L`. Alternatively, they can be obtained using the [ALSA C library](#).

Sample IDs are `hw:1,0` and `hw:CARD=CC,DEV=0`.

Audio device IDs on macOS

The following function implemented in Objective-C creates a list of the names and IDs of the audio devices attached to a Mac.

The `deviceUID` string is used to identify a device in the Speech SDK for macOS.

```

#import <Foundation/Foundation.h>
#import <CoreAudio/CoreAudio.h>

CFArrrayRef CreateInputDeviceArray()
{
    AudioObjectPropertyAddress propertyAddress = {
        kAudioHardwarePropertyDevices,
        kAudioObjectPropertyScopeGlobal,
        kAudioObjectPropertyElementMaster
    };

    UInt32 dataSize = 0;
    OSStatus status = AudioObjectGetPropertyDataSize(kAudioObjectSystemObject, &propertyAddress, 0, NULL,
&dataSize);
    if (kAudioHardwareNoError != status) {
        fprintf(stderr, "AudioObjectGetPropertyDataSize (kAudioHardwarePropertyDevices) failed: %i\n",
status);
        return NULL;
    }

    UInt32 deviceCount = (uint32)(dataSize / sizeof(AudioDeviceID));

    AudioDeviceID *audioDevices = (AudioDeviceID *)malloc(dataSize));
    if (NULL == audioDevices) {
        fputs("Unable to allocate memory", stderr);
        return NULL;
    }
}

```

```

        }

        status = AudioObjectGetPropertyData(kAudioObjectSystemObject, &propertyAddress, 0, NULL, &dataSize,
audioDevices);
        if (kAudioHardwareNoError != status) {
            fprintf(stderr, "AudioObjectGetPropertyData (kAudioHardwarePropertyDevices) failed: %i\n", status);
            free(audioDevices);
            audioDevices = NULL;
            return NULL;
        }

        CFMutableArrayRef inputDeviceArray = CFArrayCreateMutable(kCFAllocatorDefault, deviceCount,
&kCFTypeArrayCallBacks);
        if (NULL == inputDeviceArray) {
            fputs("CFArrayCreateMutable failed", stderr);
            free(audioDevices);
            audioDevices = NULL;
            return NULL;
        }

        // Iterate through all the devices and determine which are input-capable
        propertyAddress.mScope = kAudioDevicePropertyScopeInput;
        for (UInt32 i = 0; i < deviceCount; ++i) {
            // Query device UID
            CFStringRef deviceUID = NULL;
            dataSize = sizeof(deviceUID);
            propertyAddress.mSelector = kAudioDevicePropertyDeviceUID;
            status = AudioObjectGetPropertyData(audioDevices[i], &propertyAddress, 0, NULL, &dataSize,
&deviceUID);
            if (kAudioHardwareNoError != status) {
                fprintf(stderr, "AudioObjectGetPropertyData (kAudioDevicePropertyDeviceUID) failed: %i\n",
status);
                continue;
            }

            // Query device name
            CFStringRef deviceName = NULL;
            dataSize = sizeof(deviceName);
            propertyAddress.mSelector = kAudioDevicePropertyDeviceNameCFString;
            status = AudioObjectGetPropertyData(audioDevices[i], &propertyAddress, 0, NULL, &dataSize,
&deviceName);
            if (kAudioHardwareNoError != status) {
                fprintf(stderr, "AudioObjectGetPropertyData (kAudioDevicePropertyDeviceNameCFString) failed:
%i\n", status);
                continue;
            }

            // Determine if the device is an input device (it is an input device if it has input channels)
            dataSize = 0;
            propertyAddress.mSelector = kAudioDevicePropertyStreamConfiguration;
            status = AudioObjectGetPropertyDataSize(audioDevices[i], &propertyAddress, 0, NULL, &dataSize);
            if (kAudioHardwareNoError != status) {
                fprintf(stderr, "AudioObjectGetPropertyDataSize (kAudioDevicePropertyStreamConfiguration) failed:
%i\n", status);
                continue;
            }

            AudioBufferList *bufferList = (AudioBufferList *)malloc(dataSize));
            if (NULL == bufferList) {
                fputs("Unable to allocate memory", stderr);
                break;
            }

            status = AudioObjectGetPropertyData(audioDevices[i], &propertyAddress, 0, NULL, &dataSize,
bufferList);
            if (kAudioHardwareNoError != status || 0 == bufferList->mNumberBuffers) {
                if (kAudioHardwareNoError != status)
                    fprintf(stderr, "AudioObjectGetPropertyData (kAudioDevicePropertyStreamConfiguration) failed:
%i\n", status);
                free(bufferList);
            }
        }
    }
}

```

```

    free(bufferList);
    bufferList = NULL;
    continue;
}

free(bufferList);
bufferList = NULL;

// Add a dictionary for this device to the array of input devices
CFStringRef keys [] = { CFSTR("deviceUID"), CFSTR("deviceName")};
CFStringRef values [] = { deviceUID, deviceName};

CFDictionaryRef deviceDictionary = CFDictionaryCreate(kCFAllocatorDefault,
                                                       (const void **)(keys),
                                                       (const void **)(values),
                                                       2,
                                                       &kCFTypeDictionaryKeyCallBacks,
                                                       &kCFTypeDictionaryValueCallBacks);

CFArrayAppendValue(inputDeviceArray, deviceDictionary);

CFRelease(deviceDictionary);
deviceDictionary = NULL;
}

free(audioDevices);
audioDevices = NULL;

// Return a non-mutable copy of the array
CFArrayRef immutableInputDeviceArray = CFArrayCreateCopy(kCFAllocatorDefault, inputDeviceArray);
CFRelease(inputDeviceArray);
inputDeviceArray = NULL;

return immutableInputDeviceArray;
}

```

For example, the UID for the built-in microphone is `BuiltInMicrophoneDevice`.

Audio device IDs on iOS

Audio device selection with the Speech SDK is not supported on iOS. However, apps using the SDK can influence audio routing through the `AVAudioSession` Framework.

For example, the instruction

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryRecord
                                       withOptions:AVAudioSessionCategoryOptionAllowBluetooth error:NULL];
```

enables the use of a Bluetooth headset for a speech-enabled app.

Audio device IDs in JavaScript

In JavaScript the `MediaDevices.enumerateDevices()` method can be used to enumerate the media devices and find a device ID to pass to `fromMicrophone(...)`.

Next steps

[Explore our samples on GitHub](#)

See also

- [Customize acoustic models](#)
- [Customize language models](#)

Automatic language detection for speech to text

11/14/2019 • 2 minutes to read • [Edit Online](#)

Automatic language detection is used to determine the most likely match for audio passed to the Speech SDK when compared against a list of provided languages. The value returned by automatic language detection is then used to select the language model for speech to text, providing you with a more accurate transcription. To see which languages are available, see [Language support](#).

In this article, you'll learn how to use `AutoDetectSourceLanguageConfig` to construct a `SpeechRecognizer` object and retrieve the detected language.

IMPORTANT

This feature is only available for the Speech SDK for C++ and the Speech SDK for Java.

Automatic language detection with the Speech SDK

Automatic language detection currently has a services-side limit of two languages per detection. Keep this limitation in mind when construction your `AutoDetectSourceLanguageConfig` object. In the samples below, you'll create an `AutoDetectSourceLanguageConfig`, then use it to construct a `SpeechRecognizer`.

TIP

You can also specify a custom model to use when performing speech to text. For more information, see [Use a custom model for automatic language detection](#).

The following snippets illustrate how to use automatic language detection in your apps:

```
auto autoDetectSourceLanguageConfig = AutoDetectSourceLanguageConfig::FromLanguages({ "en-US", "de-DE" });
auto recognizer = SpeechRecognizer::FromConfig(speechConfig, autoDetectSourceLanguageConfig, audioConfig);
speechRecognitionResult = recognizer->RecognizeOnceAsync().get();
auto autoDetectSourceLanguageResult = AutoDetectSourceLanguageResult::FromResult(speechRecognitionResult);
auto detectedLanguage = autoDetectSourceLanguageResult->Language;
```

```
AutoDetectSourceLanguageConfig autoDetectSourceLanguageConfig =
AutoDetectSourceLanguageConfig.fromLanguages(Arrays.asList("en-US", "de-DE"));
SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, autoDetectSourceLanguageConfig, audioConfig);
Future<SpeechRecognitionResult> future = recognizer.recognizeOnceAsync();
SpeechRecognitionResult result = future.get(30, TimeUnit.SECONDS);
AutoDetectSourceLanguageResult autoDetectSourceLanguageResult =
AutoDetectSourceLanguageResult.fromResult(result);
String detectedLanguage = autoDetectSourceLanguageResult.getLanguage();

recognizer.close();
speechConfig.close();
autoDetectSourceLanguageConfig.close();
audioConfig.close();
result.close();
```

Use a custom model for automatic language detection

In addition to language detection using Speech service models, you can specify a custom model for enhanced recognition. If a custom model isn't provided, the service will use the default language model.

The snippets below illustrate how to specify a custom model in your call to the Speech service. If the detected language is `en-US`, then the default model is used. If the detected language is `fr-FR`, then the endpoint for the custom model is used:

```
std::vector<std::shared_ptr<SourceLanguageConfig>> sourceLanguageConfigs;
sourceLanguageConfigs.push_back(SourceLanguageConfig::FromLanguage("en-US"));
sourceLanguageConfigs.push_back(SourceLanguageConfig::FromLanguage("fr-FR", "The Endpoint Id for custom model
of fr-FR"));
auto autoDetectSourceLanguageConfig =
AutoDetectSourceLanguageConfig::FromSourceLanguageConfigs(sourceLanguageConfigs);
```

```
List sourceLanguageConfigs = new ArrayList<SourceLanguageConfig>();
sourceLanguageConfigs.add(SourceLanguageConfig.fromLanguage("en-US"));
sourceLanguageConfigs.add(SourceLanguageConfig.fromLanguage("fr-FR", "The Endpoint Id for custom model of fr-
FR"));
AutoDetectSourceLanguageConfig autoDetectSourceLanguageConfig =
AutoDetectSourceLanguageConfig.fromSourceLanguageConfigs(sourceLanguageConfigs);
```

Next steps

- [Speech SDK reference documentation](#)

Using codec compressed audio input with the Speech SDK

12/4/2019 • 2 minutes to read • [Edit Online](#)

The Speech SDK's **Compressed Audio Input Stream** API provides a way to stream compressed audio to the Speech service using PullStream or PushStream.

IMPORTANT

Streaming compressed input audio is currently supported for C++, C#, and Java on Linux (Ubuntu 16.04, Ubuntu 18.04, Debian 9). It is also supported for [Java in Android](#) and [Objective-C in iOS](#) platform. Speech SDK version 1.7.0 or higher is required.

For wav/PCM see the mainline speech documentation. Outside of wav/PCM, the following codec compressed input formats are supported:

- MP3
- OPUS/OGG
- FLAC
- ALAW in wav container
- MULAW in wav container

Prerequisites

Handling compressed audio is implemented using [GStreamer](#). For licensing reason Gstreamer binaries are not compiled and linked with speech SDK. So application developer needs to install the following on 18.04, 16.04 and Debian 9 to use compressed input audio.

```
sudo apt install libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly
```

Example code using codec compressed audio input

To stream in a compressed audio format to the Speech service, create `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying the compression format of the stream.

Let's assume that you have an input stream class called `myPushStream` and are using OPUS/OGG. Your code may look like this:

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;

var speechConfig = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

// Create an audio config specifying the compressed audio format and the instance of your input stream class.
var audioFormat = AudioStreamFormat.GetCompressedFormat(AudioStreamContainerFormat.OGG_OPUS);
var audioConfig = AudioConfig.FromStreamInput(myPushStream, audioFormat);

var recognizer = new SpeechRecognizer(speechConfig, audioConfig);

var result = await recognizer.RecognizeOnceAsync();

var text = result.GetText();
```

Next steps

- [Get your Speech trial subscription](#)
- [See how to recognize speech in Java](#)

How to: Use codec compressed audio input with the Speech SDK on iOS

12/4/2019 • 2 minutes to read • [Edit Online](#)

The Speech SDK's **Compressed Audio Input Stream** API provides a way to stream compressed audio to the Speech service using a pull or push stream.

IMPORTANT

Speech SDK version 1.7.0 or higher is required for streaming compressed audio on iOS. It is also supported for [C++](#), [C#](#), and [Java on Linux \(Ubuntu 16.04, Ubuntu 18.04, Debian 9\)](#) and [Java in Android](#).

For wav/PCM see the mainline speech documentation. Outside of wav/PCM, the following codec compressed input formats are supported:

- MP3
- OPUS/OGG
- FLAC
- ALAW in wav container
- MULAW in wav container

Prerequisites

Handling compressed audio is implemented using [GStreamer](#). For licensing reasons, these functions can not be shipped with the SDK, but a wrapper library containing these functions needs to be built by application developers and shipped with the apps using the SDK.

To build this wrapper library, first download and install the [GStreamer SDK](#). Then, download the Xcode project for the [wrapper library](#).

Open the project in Xcode and build it for the **Generic iOS Device** target -- it will not work to build it for a specific target.

The build step will generate a dynamic framework bundle with a dynamic library for all necessary architectures with the name of `GStreamerWrapper.framework`.

This framework must be included in all apps that use compressed audio streams with the Speech service SDK.

Apply the following settings in your Xcode project to accomplish this:

1. Copy both the `GStreamerWrapper.framework` you just built and the framework of the Cognitive Services Speech SDK, which you can download from [here](#), to the directory containing your sample project.
2. Adjust the paths to the frameworks in the *Project Settings*.
 - a. In the **General** tab under the **Embedded Binaries** header, add the SDK library as a framework: **Add embedded binaries** > **Add other...** > Navigate to the directory you chose and select both frameworks.
 - b. Go to the **Build Settings** tab and activate **All** settings.
3. Add the directory `$(SRCROOT)/...` to the *Framework Search Paths* under the **Search Paths** heading.

Example code using codec compressed audio input

To stream in a compressed audio format to the Speech service, create a `SPXPullAudioInputStream` or `SPXPushAudioInputStream`.

The following snippet shows how to create an `SPXAudioConfiguration` from an instance of a `SPXPushAudioInputStream`, specifying mp3 as the compression format of the stream.

```
// <setup-stream>
SPXAudioStreamContainerFormat compressedStreamFormat = SPXAudioStreamContainerFormat_MP3;
SPXAudioStreamFormat *audioFormat = [[SPXAudioStreamFormat alloc]
initUsingCompressedFormat:compressedStreamFormat];
SPXPushAudioInputStream* stream = [[SPXPushAudioInputStream alloc] initWithAudioFormat:audioFormat];

SPXAudioConfiguration* audioConfig = [[SPXAudioConfiguration alloc] initWithStreamInput:stream];
if (!audioConfig) {
    NSLog(@"Error creating stream!");
    [self updateRecognitionErrorText:(@"Error creating stream!")];
    return;
}
// </setup-stream>
```

The next snippet shows how compressed audio data can be read from a file and pumped into the `SPXPushAudioInputStream`.

```

// <push-compressed-stream>
NSInputStream *compressedStream = [[NSInputStream alloc] initWithFileAtPath:weatherFile];
[compressedStream open];
NSLog(@"result of opening stream: %@", compressedStream.streamError, (unsigned
long)compressedStream.streamStatus);
if (nil == compressedStream)
{
    NSLog(@"Error while opening file");
    audioConfig = nil;
    return;
}

// start recognizing
[self updateRecognitionStatusText:(@"Recognizing from push stream...")];
[speechRecognizer startContinuousRecognition];

const NSInteger nBytesToRead = 1000;
// push data to stream;
uint8_t *buffer = malloc(nBytesToRead);
NSInteger nBytesRead = 0;
while (1)
{
    // read data
    nBytesRead = [compressedStream read:buffer maxLength:nBytesToRead];
    if (0 == nBytesRead) {
        NSLog(@"end of stream reached");
        [stream close];
        break;
    }
    else if (0 > nBytesRead) {
        NSLog(@"error reading stream (%ld): %@", (long)nBytesRead, compressedStream.streamError,
compressedStream.streamStatus);
        [stream close];
        break;
    }
    else
    {
        NSLog(@"Read %lu bytes from file", nBytesRead);
        NSData *data = [NSData dataWithBytesNoCopy:buffer length:nBytesRead freeWhenDone:NO];

        [stream write:data];
        NSLog(@"Wrote %lu bytes to stream", [data length]);
    }
    [NSThread sleepForTimeInterval:0.1f];
}

[speechRecognizer stopContinuousRecognition];
// </push-compressed-stream>

```

Next steps

- [Get your Speech trial subscription](#)
- [See how to recognize speech in Java](#)

How to: Use codec compressed audio input with the Speech SDK on Android

12/4/2019 • 3 minutes to read • [Edit Online](#)

The Speech SDK's **Compressed Audio Input Stream** API provides a way to stream compressed audio to the Speech service using PullStream or PushStream.

IMPORTANT

Streaming compressed input audio is currently supported for **C++**, **C#**, and **Java** on Linux (**Ubuntu 16.04**, **Ubuntu 18.04**, **Debian 9**). It is also supported for Java in Android and **Objective-C** in **iOS** platform. Speech SDK version 1.7.0 or higher is required.

For wav/PCM see the mainline speech documentation. Outside of wav/PCM, the following codec compressed input formats are supported:

- MP3
- OPUS/OGG
- FLAC
- ALAW in wav container
- MULAW in wav container

Prerequisites to using codec compressed audio input on Android

Codec compressed audio is implemented using **GStreamer**. For licensing reasons, Gstreamer binaries are not compiled with the SDK. You'll need to use the prebuilt binaries for Android. To download the prebuilt libraries, see [Installing for Android Development](#).

`libgstreamer_android.so` is required. Make sure that your GStreamer plugins are linked in `libgstreamer_android.so`.

```
GSTREAMER_PLUGINS := coreelements app audioconvert mpg123 audioresample audioparsers ogg opusparse opus
wvparse alaw mulaw flac
```

An example `Android.mk` and `Application.mk` file are provided below. Follow these steps to create the gstreamer shared object: `libgstreamer_android.so`.

```

# Android.mk
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE     := dummy
LOCAL_SHARED_LIBRARIES := gstreamer_android
LOCAL_LDLIBS := -llog
include $(BUILD_SHARED_LIBRARY)

ifndef GSTREAMER_ROOT_ANDROID
$(error GSTREAMER_ROOT_ANDROID is not defined!)
endif

ifndef APP_BUILD_SCRIPT
$(error APP_BUILD_SCRIPT is not defined!)
endif

ifndef TARGET_ARCH_ABI
$(error TARGET_ARCH_ABI is not defined!)
endif

ifeq ($(TARGET_ARCH_ABI),armeabi)
GSTREAMER_ROOT      := $(GSTREAMER_ROOT_ANDROID)/arm
else ifeq ($(TARGET_ARCH_ABI),armeabi-v7a)
GSTREAMER_ROOT      := $(GSTREAMER_ROOT_ANDROID)/armv7
else ifeq ($(TARGET_ARCH_ABI),arm64-v8a)
GSTREAMER_ROOT      := $(GSTREAMER_ROOT_ANDROID)/arm64
else ifeq ($(TARGET_ARCH_ABI),x86)
GSTREAMER_ROOT      := $(GSTREAMER_ROOT_ANDROID)/x86
else ifeq ($(TARGET_ARCH_ABI),x86_64)
GSTREAMER_ROOT      := $(GSTREAMER_ROOT_ANDROID)/x86_64
else
$(error Target arch ABI not supported: $(TARGET_ARCH_ABI))
endif

GSTREAMER_NDK_BUILD_PATH := $(GSTREAMER_ROOT)/share/gst-android/ndk-build/
include $(GSTREAMER_NDK_BUILD_PATH)/plugins.mk
GSTREAMER_PLUGINS      := coreelements app audioconvert mpg123 audioresample audioparsers ogg opusparse
opus wavparse alaw mulaw flac
GSTREAMER_EXTRA_LIBS    := -liconv
include $(GSTREAMER_NDK_BUILD_PATH)/gstreamer-1.0.mk

```

```

# Application.mk
APP_STL = c++_shared
APP_PLATFORM = android-21
APP_BUILD_SCRIPT = Android.mk

```

You can build `libgstreamer_android.so` using the following command on Ubuntu 16.04 or 18.04. The following command lines have only been tested for [Gstreamer Android version 1.14.4](#) with [Android NDK b16b](#).

```

# assuming wget and unzip already installed on the system
mkdir buildLibGstreamer
cd buildLibGstreamer
wget https://dl.google.com/android/repository/android-ndk-r16b-linux-x86_64.zip
unzip -q -o android-ndk-r16b-linux-x86_64.zip
export PATH=$PATH:$(pwd)/android-ndk-r16b
export NDK_PROJECT_PATH=$(pwd)/android-ndk-r16b
wget https://gstreamer.freedesktop.org/data/pkg/android/1.14.4/gstreamer-1.0-android-universal-1.14.4.tar.bz2
mkdir gstreamer_android
tar -xjf gstreamer-1.0-android-universal-1.14.4.tar.bz2 -C $(pwd)/gstreamer_android/
export GSTREAMER_ROOT_ANDROID=$(pwd)/gstreamer_android

mkdir gstreamer
# Copy the Application.mk and Android.mk from the documentation above and put it inside $(pwd)/gstreamer

# Enable only one of the following at one time to create the shared object for the targeted ABI
echo "building for armeabi-v7a. libgstreamer_android.so will be placed in $(pwd)/armeabi-v7a"
ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=armeabi-v7a NDK_LIBS_OUT=$(pwd)

#echo "building for arm64-v8a. libgstreamer_android.so will be placed in $(pwd)/arm64-v8a"
#ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=arm64-v8a NDK_LIBS_OUT=$(pwd)

#echo "building for x86_64. libgstreamer_android.so will be placed in $(pwd)/x86_64"
#ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=x86_64 NDK_LIBS_OUT=$(pwd)

#echo "building for x86. libgstreamer_android.so will be placed in $(pwd)/x86"
#ndk-build -C $(pwd)/gstreamer "NDK_APPLICATION_MK=Application.mk" APP_ABI=x86 NDK_LIBS_OUT=$(pwd)

```

Once the shared object (libgstreamer_android.so) is built application developer needs to place the shared object in the Android app, so that it can be loaded by speech SDK.

Example code using codec compressed audio input

To stream in a compressed audio format to the Speech service, create `PullAudioInputStream` or `PushAudioInputStream`. Then, create an `AudioConfig` from an instance of your stream class, specifying the compression format of the stream.

Let's assume that you have an input stream class called `myPullStream` and are using OPUS/OGG. Your code may look like this:

```

import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import com.microsoft.cognitiveservices.speech.audio.AudioInputStream;
import com.microsoft.cognitiveservices.speech.audio.AudioStreamFormat;
import com.microsoft.cognitiveservices.speech.audio.PullAudioInputStream;
import com.microsoft.cognitiveservices.speech.internal.AudioStreamContainerFormat;

SpeechConfig speechConfig = SpeechConfig.fromSubscription("YourSubscriptionKey", "YourServiceRegion");

// Create an audio config specifying the compressed audio format and the instance of your input stream class.
AudioStreamFormat audioFormat = AudioStreamFormat.getCompressedFormat(AudioStreamContainerFormat.OGG_OPUS);
AudioConfig audioConfig = AudioConfig.fromStreamInput(myPullStream, audioFormat);

SpeechRecognizer recognizer = new SpeechRecognizer(speechConfig, audioConfig);

SpeechRecognitionResult result = recognizer.recognizeOnceAsync().get()

String text = result.getText();

```

Next steps

- [Get your Speech trial subscription](#)

- See how to recognize speech in Java

Install and run Speech service containers (Preview)

12/4/2019 • 17 minutes to read • [Edit Online](#)

Containers enable you to run some of the Speech service APIs in your own environment. Containers are great for specific security and data governance requirements. In this article you'll learn how to download, install, and run a Speech container.

Speech containers enable customers to build a speech application architecture that is optimized for both robust cloud capabilities and edge locality. There are four different containers available. The two standard containers are **Speech-to-text** and **Text-to-speech**. The two custom containers are **Custom Speech-to-text** and **Custom Text-to-speech**.

IMPORTANT

All speech containers are currently offered as part of a [Public "Gated" Preview](#). An announcement will be made when speech containers progress to General Availability (GA).

FUNCTION	FEATURES	LATEST
Speech-to-text	Transcribes continuous real-time speech or batch audio recordings into text with intermediate results.	2.0.0
Custom Speech-to-text	Using a custom model from the Custom Speech portal , transcribes continuous real-time speech or batch audio recordings into text with intermediate results.	2.0.0
Text-to-speech	Converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.3.0
Custom Text-to-speech	Using a custom model from the Custom Voice portal , converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.3.0

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

The following prerequisites before using Speech containers:

REQUIRED	PURPOSE
----------	---------

REQUIRED	PURPOSE
Docker Engine	<p>You need the Docker Engine installed on a host computer. Docker provides packages that configure the Docker environment on macOS, Windows, and Linux. For a primer on Docker and container basics, see the Docker overview.</p> <p>Docker must be configured to allow the containers to connect with and send billing data to Azure.</p> <p>On Windows, Docker must also be configured to support Linux containers.</p>
Familiarity with Docker	<p>You should have a basic understanding of Docker concepts, like registries, repositories, containers, and container images, as well as knowledge of basic <code>docker</code> commands.</p>
Speech resource	<p>In order to use these containers, you must have:</p> <p>An Azure <i>Speech</i> resource to get the associated API key and endpoint URI. Both values are available on the Azure portal's Speech Overview and Keys pages. They are both required to start the container.</p> <p>{API_KEY}: One of the two available resource keys on the Keys page</p> <p>{ENDPOINT_URI}: The endpoint as provided on the Overview page</p>

Request access to the container registry

Fill out and submit the [Cognitive Services Speech Containers Request form](#) to request access to the container.

The form requests information about you, your company, and the user scenario for which you'll use the container. After you've submitted the form, the Azure Cognitive Services team reviews it to ensure that you meet the criteria for access to the private container registry.

IMPORTANT

You must use an email address that's associated with either a Microsoft Account (MSA) or Azure Active Directory (Azure AD) account in the form.

If your request is approved, you'll receive an email with instructions that describe how to obtain your credentials and access the private container registry.

Use the Docker CLI to authenticate the private container registry

You can authenticate with the private container registry for Cognitive Services Containers in any of several ways, but the recommended method from the command line is to use the [Docker CLI](#).

Use the `docker login` command, as shown in the following example, to log in to `containerpreview.azurecr.io`, the private container registry for Cognitive Services Containers. Replace `<username>` with the user name and `<password>` with the password that's provided in the credentials you received from the Azure Cognitive Services team.

```
docker login containerpreview.azurecr.io -u <username> -p <password>
```

If you've secured your credentials in a text file, you can concatenate the contents of that text file, by using the `cat` command, to the `docker login` command, as shown in the following example. Replace `<passwordFile>` with the path and name of the text file that contains the password and `<username>` with the user name that's provided in your credentials.

```
cat <passwordFile> | docker login containerpreview.azurecr.io -u <username> --password-stdin
```

Gathering required parameters

There are three primary parameters for all Cognitive Services' containers that are required. The end-user license agreement (EULA) must be present with a value of `accept`. Additionally, both an Endpoint URL and API Key are needed.

Endpoint URI {ENDPOINT_URI}

The **Endpoint** URI value is available on the Azure portal *Overview* page of the corresponding Cognitive Service resource. Navigate to the *Overview* page, hover over the Endpoint, and a `Copy to clipboard` icon will appear. Copy and use where needed.

Home > widgets

wIDGETS Cognitive Services

Search (Ctrl+ /)

Overview Copy to clipboard

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

RESOURCE MANAGEMENT

Keys Copy to clipboard

Quick start

Pricing tier

Billing By Subscription

Resource group (change)
widgets-resource-group

Status
Active

Location
North Central US

Subscription (change)
widgets-subscription

Subscription ID
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Tags (change)
Click here to add tags

API type
<API Type>

Pricing tier
Standard

Endpoint
<https://widgets.cognitiveservices.azure.com/api/example-endpoint>

Manage keys
Show access keys ...

Keys {API_KEY}

This key is used to start the container, and is available on the Azure portal's Keys page of the corresponding Cognitive Service resource. Navigate to the *Keys* page, and click on the `Copy to clipboard` icon.

Home > widgets - Keys

wIDGETS - Keys Cognitive Services

Search (Ctrl+ /)

Regenerate Key1 Regenerate Key2

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

RESOURCE MANAGEMENT

Keys Copy to clipboard

Quick start

Pricing tier

Billing By Subscription

NAME
widgets

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

KEY 1
<key 1 value> Copy to clipboard

KEY 2
<key 2 value> Copy to clipboard

IMPORTANT

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely, for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

The host computer

The host is a x64-based computer that runs the Docker container. It can be a computer on your premises or a Docker hosting service in Azure, such as:

- [Azure Kubernetes Service](#).
- [Azure Container Instances](#).
- A [Kubernetes](#) cluster deployed to [Azure Stack](#). For more information, see [Deploy Kubernetes to Azure Stack](#).

Advanced Vector Extension support

The **host** is the computer that runs the docker container. The host *must support* [Advanced Vector Extensions](#) (AVX2). You can check for AVX2 support on Linux hosts with the following command:

```
grep -q avx2 /proc/cpuinfo && echo AVX2 supported || echo No AVX2 support detected
```

WARNING

The host computer is *required* to support AVX2. The container *will not* function correctly without AVX2 support.

Container requirements and recommendations

The following table describes the minimum and recommended allocation of resources for each Speech container.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

CONTAINER	MINIMUM	RECOMMENDED
Speech-to-text	2 core, 2-GB memory	4 core, 4-GB memory

- Each core must be at least 2.6 gigahertz (GHz) or faster.

Core and memory correspond to the `--cpus` and `--memory` settings, which are used as part of the `docker run` command.

NOTE

The minimum and recommended are based off of Docker limits, *not* the host machine resources. For example, speech-to-text containers memory map portions of a large language model, and it is *recommended* that the entire file fits in memory, which is an additional 4-6 GB. Also, the first run of either container may take longer, since models are being paged into memory.

Get the container image with `docker pull`

Container images for Speech are available in the following Container Registry.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

CONTAINER	REPOSITORY
Speech-to-text	containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text:latest

TIP

You can use the [docker images](#) command to list your downloaded container images. For example, the following command lists the ID, repository, and tag of each downloaded container image, formatted as a table:

```
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
IMAGE ID          REPOSITORY          TAG
<image-id>      <repository-path/name>  <tag-name>
```

Docker pull for the Speech containers

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

Docker pull for the Speech-to-text container

Use the [docker pull](#) command to download a container image from Container Preview registry.

```
docker pull containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text:latest
```

IMPORTANT

The `latest` tag pulls the `en-us` locale. For additional locales see [Speech-to-text locales](#).

Speech-to-text locales

All tags, except for `latest` are in the following format and are case-sensitive:

```
<major>.<minor>.<patch>-<platform>-<locale>-<prerelease>
```

The following tag is an example of the format:

```
2.0.0-amd64-en-us-preview
```

For all of the supported locales of the **speech-to-text** container, please see [Speech-to-text image tags](#).

How to use the container

Once the container is on the [host computer](#), use the following process to work with the container.

1. [Run the container](#), with the required billing settings. More [examples](#) of the `docker run` command are available.

2. Query the container's prediction endpoint

Run the container with `docker run`

Use the `docker run` command to run the container. Refer to [gathering required parameters](#) for details on how to get the `{Endpoint_URI}` and `{API_Key}` values. Additional [examples](#) of the `docker run` command are also available.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

To run the *Speech-to-text* container, execute the following `docker run` command.

```
docker run --rm -it -p 5000:5000 --memory 4g --cpus 4 \
containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

This command:

- Runs a *Speech-to-text* container from the container image.
- Allocates 4 CPU cores and 4 gigabytes (GB) of memory.
- Exposes TCP port 5000 and allocates a pseudo-TTY for the container.
- Automatically removes the container after it exits. The container image is still available on the host computer.

IMPORTANT

The `Eula`, `Billing`, and `ApiKey` options must be specified to run the container; otherwise, the container won't start. For more information, see [Billing](#).

Query the container's prediction endpoint

CONTAINERS	SDK HOST URL	PROTOCOL
Speech-to-text and Custom Speech-to-text	<code>ws://localhost:5000</code>	WS
Text-to-speech and Custom Text-to-speech	<code>http://localhost:5000</code>	HTTP

For more information on using WSS and HTTPS protocols, see [container security](#).

Speech-to-text or Custom Speech-to-text

The container provides websocket-based query endpoint APIs, that are accessed through the [Speech SDK](#). By default, the Speech SDK uses online speech services. To use the container, you need to change the initialization method.

TIP

When using the Speech SDK with containers, you do not need to provide the Azure Speech resource [subscription key](#) or an [authentication bearer token](#).

See the examples below.

- C#
- Python

Change from using this Azure-cloud initialization call:

```
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

to this call using the container **host**:

```
var config = SpeechConfig.FromHost(  
    new Uri("ws://localhost:5000"));
```

Text-to-speech or Custom Text-to-speech

The container provides [REST-based endpoint APIs](#). There are many [sample source code projects](#) for platform, framework, and language variations available.

With the *Standard Text-to-speech* container, you should rely on the locale and voice of the image tag you downloaded. For example, if you downloaded the `latest` tag the default locale is `en-US` and the `JessaRUS` voice. The `{VOICE_NAME}` argument would then be `en-US-JessaRUS`. See the example SSML below:

```
<speak version="1.0" xml:lang="en-US">  
  <voice name="en-US-JessaRUS">  
    This text will get converted into synthesized speech.  
  </voice>  
</speak>
```

However, for *Custom Text-to-speech* you'll need to obtain the **Voice / model** from the [custom voice portal](#). The custom model name is synonymous with the voice name. Navigate to the **Training** page, and copy the **Voice / model** to use as the `{VOICE_NAME}` argument.

Text input ↑	Voice / model ↑	Created ↓	Status ↑↓	Audio
Hi. this is my custom voice.	custom-voice-model	10/15/2019 7:29 AM	Succeeded	[Speaker icon]

See the example SSML below:

```
<speak version="1.0" xml:lang="en-US">  
  <voice name="custom-voice-model">  
    This text will get converted into synthesized speech.  
  </voice>  
</speak>
```

Let's construct an HTTP POST request, providing a few headers and a data payload. Replace the `{VOICE_NAME}`

placeholder with your own value.

```
curl -s -v -X POST http://localhost:5000/speech/synthesize/cognitiveservices/v1 \
-H 'Accept: audio/*' \
-H 'Content-Type: application/ssml+xml' \
-H 'X-Microsoft-OutputFormat: riff-16khz-16bit-mono-pcm' \
-d '<speak version="1.0" xml:lang="en-US"><voice name="{VOICE_NAME}">This is a test, only a test.</voice>
</speak>'
```

This command:

- Constructs an HTTP POST request for the `speech/synthesize/cognitiveservices/v1` endpoint.
- Specifies an `Accept` header of `audio/*`
- Specifies a `Content-Type` header of `application/ssml+xml`, for more information, see [request body](#).
- Specifies a `X-Microsoft-OutputFormat` header of `riff-16khz-16bit-mono-pcm`, for more options see [audio output](#).
- Sends the [Speech Synthesis Markup Language \(SSML\)](#) request given the `{VOICE_NAME}` to the endpoint.

Run multiple containers on the same host

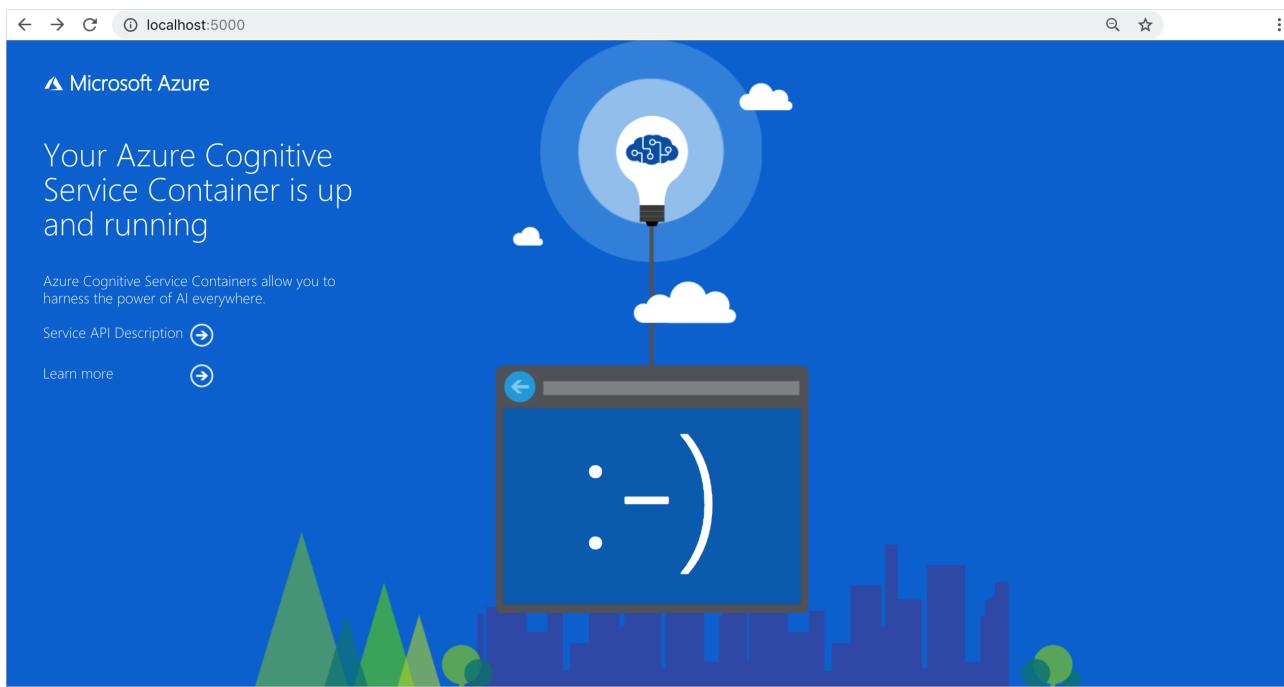
If you intend to run multiple containers with exposed ports, make sure to run each container with a different exposed port. For example, run the first container on port 5000 and the second container on port 5001.

You can have this container and a different Azure Cognitive Services container running on the HOST together. You also can have multiple containers of the same Cognitive Services container running.

Validate that a container is running

There are several ways to validate that the container is running. Locate the *External IP* address and exposed port of the container in question, and open your favorite web browser. Use the various request URLs below to validate the container is running. The example request URLs listed below are `http://localhost:5000`, but your specific container may vary. Keep in mind that you're to rely on your container's *External IP* address and exposed port.

REQUEST URL	PURPOSE
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/status</code>	Requested with an HTTP GET, to validate that the container is running without causing an endpoint query. This request can be used for Kubernetes liveness and readiness probes .
<code>http://localhost:5000/swagger</code>	The container provides a full set of documentation for the endpoints and a Try it out feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.



Stop the container

To shut down the container, in the command-line environment where the container is running, select **Ctrl+C**.

Troubleshooting

When starting or running the container, you may experience issues. Use an output **mount** and enable logging. Doing so will allow the container to generate log files that are helpful when troubleshooting issues.

TIP

For more troubleshooting information and guidance, see [Cognitive Services containers frequently asked questions \(FAQ\)](#).

Billing

The Speech containers send billing information to Azure, using a *Speech* resource on your Azure account.

Queries to the container are billed at the pricing tier of the Azure resource that's used for the `<ApiKey>`.

Azure Cognitive Services containers aren't licensed to run without being connected to the billing endpoint for metering. You must enable the containers to communicate billing information with the billing endpoint at all times. Cognitive Services containers don't send customer data, such as the image or text that's being analyzed, to Microsoft.

Connect to Azure

The container needs the billing argument values to run. These values allow the container to connect to the billing endpoint. The container reports usage about every 10 to 15 minutes. If the container doesn't connect to Azure within the allowed time window, the container continues to run but doesn't serve queries until the billing endpoint is restored. The connection is attempted 10 times at the same time interval of 10 to 15 minutes. If it can't connect to the billing endpoint within the 10 tries, the container stops running.

Billing arguments

For the `docker run` command to start the container, all three of the following options must be specified with valid values:

OPTION	DESCRIPTION
ApiKey	The API key of the Cognitive Services resource that's used to track billing information. The value of this option must be set to an API key for the provisioned resource that's specified in <code>Billing</code> .
Billing	The endpoint of the Cognitive Services resource that's used to track billing information. The value of this option must be set to the endpoint URI of a provisioned Azure resource.
Eula	Indicates that you accepted the license for the container. The value of this option must be set to <code>accept</code> .

For more information about these options, see [Configure containers](#).

Blog posts

- [Running Cognitive Services Containers](#)
- [Azure Cognitive Services](#)

Developer samples

Developer samples are available at our [GitHub repository](#).

View webinar

Join the [webinar](#) to learn about:

- How to deploy Cognitive Services to any machine using Docker
- How to deploy Cognitive Services to AKS

Summary

In this article, you learned concepts and workflow for downloading, installing, and running Speech containers. In summary:

- Speech provides four Linux containers for Docker, encapsulating various capabilities:
 - *Speech-to-text*
 - *Custom Speech-to-text*
 - *Text-to-speech*
 - *Custom Text-to-speech*
- Container images are downloaded from the container registry in Azure.
- Container images run in Docker.
- You can use either the REST API or SDK to call operations in Speech containers by specifying the host URL of the container.
- You're required to provide billing information when instantiating a container.

IMPORTANT

Cognitive Services containers are not licensed to run without being connected to Azure for metering. Customers need to enable the containers to communicate billing information with the metering service at all times. Cognitive Services containers do not send customer data (e.g., the image or text that is being analyzed) to Microsoft.

Next steps

- Review [configure containers](#) for configuration settings
- Learn how to [use Speech service containers with Kubernetes and Helm](#)
- Use more [Cognitive Services containers](#)

Release notes

12/16/2019 • 14 minutes to read • [Edit Online](#)

Speech SDK 1.8.0: 2019-November release

New Features

- Added a `FromHost()` API, to ease use with on-prem containers and sovereign clouds.
- Added Automatic Source Language Detection for Speech Recognition (in Java and C++)
- Added `SourceLanguageConfig` object for Speech Recognition, used to specify expected source languages (in Java and C++)
- Added `KeywordRecognizer` support on Windows (UWP), Android and iOS through the Nuget and Unity packages
- Added Remote Conversation Java API to do Conversation Transcription in asynchronous batches.

Breaking changes

- Conversation Transcriber functionalities moved under namespace
`Microsoft.CognitiveServices.Speech.Transcription`.
- Part of the Conversation Transcriber methods are moved to new `Conversation` class.
- Dropped support for 32-bit (ARMv7 and x86) iOS

Bug fixes

- Fix for crash if local `KeywordRecognizer` is used without a valid speech service subscription key

Samples

- Xamarin sample for `KeywordRecognizer`
- Unity sample for `KeywordRecognizer`
- C++ and Java samples for Automatic Source Language Detection.

Speech SDK 1.7.0: 2019-September release

New Features

- Added beta support for Xamarin on Universal Windows Platform (UWP), Android, and iOS
- Added iOS support for Unity
- Added `Compressed` input support for ALaw, Mulaw, FLAC on Android, iOS and Linux
- Added `SendMessageAsync` in `Connection` class for sending a message to service
- Added `SetMessageProperty` in `Connection` class for setting property of a message
- TTS added bindings for Java (Jre and Android), Python, Swift, and Objective-C
- TTS added playback support for macOS, iOS, and Android.
- Added "word boundary" information for TTS.

Bug fixes

- Fixed IL2CPP build issue on Unity 2019 for Android
- Fixed issue with malformed headers in wav file input being processed incorrectly
- Fixed issue with UUIDs not being unique in some connection properties

- Fixed a few warnings about nullability specifiers in the Swift bindings (might require small code changes)
- Fixed a bug that caused websocket connections to be closed ungracefully under network load
- Fixed an issue on Android that sometimes results in duplicate impression IDs used by `DialogServiceConnector`
- Improvements to the stability of connections across multi-turn interactions and the reporting of failures (via `Canceled` events) when they occur with `DialogServiceConnector`
- `DialogServiceConnector` session starts will now properly provide events, including when calling `ListenOnceAsync()` during an active `StartKeywordRecognitionAsync()`
- Addressed a crash associated with `DialogServiceConnector` activities being received

Samples

- Quickstart for Xamarin
- Updated CPP Quickstart with Linux ARM64 information
- Updated Unity quickstart with iOS information

Speech SDK 1.6.0: 2019-June release

Samples

- Quickstart samples for Text To Speech on UWP and Unity
- Quickstart sample for Swift on iOS
- Unity samples for Speech & Intent Recognition and Translation
- Updated quickstart samples for `DialogServiceConnector`

Improvements / Changes

- Dialog namespace:
 - `SpeechBotConnector` has been renamed to `DialogServiceConnector`
 - `BotConfig` has been renamed to `DialogServiceConfig`
 - `BotConfig::FromChannelSecret()` has been remapped to `DialogServiceConfig::FromBotSecret()`
 - All existing Direct Line Speech clients continue to be supported after the rename
- Update TTS REST adapter to support proxy, persistent connection
- Improve error message when an invalid region is passed
- Swift/Objective-C:
 - Improved error reporting: Methods that can result in an error are now present in two versions: One that exposes an `NSError` object for error handling, and one that raises an exception. The former are exposed to Swift. This change requires adaptations to existing Swift code.
 - Improved event handling

Bug fixes

- Fix for TTS: where `SpeakTextAsync` future returned without waiting until audio has completed rendering
- Fix for marshaling strings in C# to enable full language support
- Fix for .NET core app problem to load core library with net461 target framework in samples
- Fix for occasional issues to deploy native libraries to the output folder in samples
- Fix for web socket closing reliably
- Fix for possible crash while opening a connection under very heavy load on Linux
- Fix for missing metadata in the framework bundle for macOS
- Fix for problems with `pip install --user` on Windows

Speech SDK 1.5.1

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see [here](#).
- Conversation transcription functionality is available through the SDK. See [here](#).
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `outputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the [text-to-speech overview](#).
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details [here](#)).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C

support for macOS is currently in beta.

- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.
- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details [here](#)).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing [audio input device selection](#). This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the [GitHub sample repository](#). This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our [Unity quickstart](#).
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our [sample repository](#):

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `connection` object in iOS.
- Additional Java samples for translation with audio output.

- New sample for use of the [Batch Transcription REST API](#).

Improvements / Changes

- Python
 - Improved parameter verification and error messages in `SpeechConfig`.
 - Add support for the `Connection` object.
 - Support for 32-bit Python (x86) on Windows.
 - The Speech SDK for Python is out of beta.
- iOS
 - The SDK is now built against the iOS SDK version 12.1.
 - The SDK now supports iOS versions 9.2 and later.
 - Improve reference documentation and fix several property names.
- JavaScript
 - Add support for the `Connection` object.
 - Add type definition files for bundled JavaScript
 - Initial support and implementation for phrase hints.
 - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at `turn.end`, not at `speech.end`.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
 - The Beta version of Python support (3.5 and above) is available with this release. For more information, see [here](#)](quickstart-python.md).
- JavaScript

- The Speech SDK for JavaScript has been open-sourced. The source code is available on [GitHub](#).
- We now support Node.js, more info can be found [here](#).
- The length restriction for audio sessions has been removed, reconnection will happen automatically under the cover.
- `Connection` object
 - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
 - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; `RecognizedIntent` was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the [sample repository](#).

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.

- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in `AddIntent` in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the [sample repository](#).

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our [sample repository](#), a new sample for JavaScript was added.

Cognitive Services Speech SDK 1.0.0: 2018-September release

New features

- Support for Objective-C on iOS. Check out our [Objective-C quickstart for iOS](#).
- Support for JavaScript in browser. Check out our [JavaScript quickstart](#).

Breaking changes

- With this release, a number of breaking changes are introduced. Check [this page](#) for details.

Cognitive Services Speech SDK 0.6.0: 2018-August release

New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the [UWP quickstart](#).
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the [Java Runtime](#)

[Environment quickstart](#).

Functional change

- Expose additional error detail information on connection errors.

Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.
- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

Cognitive Services Speech SDK 0.5.0: 2018-July release

New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the [Android quickstart](#).
- Support .NET Standard 2.0 on Windows. Check out the [.NET Core quickstart](#).
- Experimental: Support UWP on Windows (version 1709 or later).
 - Check out the [UWP quickstart](#).
 - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support `AuthorizationToken` for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- `SpeechOutputFormat` in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
 - In Java, the `read` method now returns `long` instead of `int`.
 - In C#, the `Read` method now returns `uint` instead of `int`.
 - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a [regions](#) page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- `AudioInputStream`

A recognizer now can consume a stream as the audio source. For more information, see the related [how-to guide](#).

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- Fired a `NoMatch` result when the response from service is timed out.
- The media foundation libraries on Windows are delay loaded. This library is required for microphone input only.
- The upload speed for audio data is limited to about twice the original audio speed.
- On Windows, C# .NET assemblies now are strong named.
- Documentation fix: `Region` is required information to create a recognizer.

More samples have been added and are constantly being updated. For the latest set of samples, see the [Speech SDK samples GitHub repository](#).

Cognitive Services Speech SDK 0.2.12733: 2018-May release

This release is the first public preview release of the Cognitive Services Speech SDK.

Speech-to-text REST API

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

As an alternative to the [Speech SDK](#), the Speech service allows you to convert speech-to-text using a REST API. Each accessible endpoint is associated with a region. Your application requires a subscription key for the endpoint you plan to use.

Before using the speech-to-text REST API, understand:

- Requests that use the REST API and transmit audio directly can only contain up to 60 seconds of audio.
- The speech-to-text REST API only returns final results. Partial results are not provided.

If sending longer audio is a requirement for your application, consider using the [Speech SDK](#) or a file-based REST API, like [batch transcription](#).

Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each service:

SUPPORTED AUTHORIZATION HEADERS	SPEECH-TO-TEXT	TEXT-TO-SPEECH
Ocp-Apim-Subscription-Key	Yes	No
Authorization: Bearer	Yes	Yes

When using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your subscription key. For example:

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your subscription key for an access token that's valid for 10 minutes. In the next few sections you'll learn how to get a token, and use a token.

How to get an access token

To get an access token, you'll need to make a request to the `issueToken` endpoint using the `Ocp-Apim-Subscription-Key` and your subscription key.

These regions and endpoints are supported:

REGION	TOKEN SERVICE ENDPOINT
Australia East	https://australiaeast.api.cognitive.microsoft.com/sts/v1.0/issueToken
Canada Central	https://canadacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken
Central US	https://centralus.api.cognitive.microsoft.com/sts/v1.0/issueToken
East Asia	https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken
East US	https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken
East US 2	https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken
France Central	https://francecentral.api.cognitive.microsoft.com/sts/v1.0/issueToken
India Central	https://centralindia.api.cognitive.microsoft.com/sts/v1.0/issueToken
Japan East	https://japaneast.api.cognitive.microsoft.com/sts/v1.0/issueToken
Korea Central	https://koreacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken
North Central US	https://northcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken
North Europe	https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken
South Central US	https://southcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken
Southeast Asia	https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken

REGION	TOKEN SERVICE ENDPOINT
UK South	https://uksouth.api.cognitive.microsoft.com/sts/v1.0/issueToken
West Europe	https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken
West US	https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
West US 2	https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken

Use these samples to create your access token request.

HTTP sample

This example is a simple HTTP request to get a token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.api.cognitive.microsoft.com
Content-type: application/x-www-form-urlencoded
Content-Length: 0
```

The body of the response contains the access token in JSON Web Token (JWT) format.

PowerShell sample

This example is a simple PowerShell script to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
$FetchTokenHeader = @{
    'Content-type'='application/x-www-form-urlencoded';
    'Content-Length'='0';
    'Ocp-Apim-Subscription-Key' = 'YOUR_SUBSCRIPTION_KEY'
}

$OAuthToken = Invoke-RestMethod -Method POST -Uri https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
-Headers $FetchTokenHeader

# show the token received
$OAuthToken
```

cURL sample

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). This cURL command illustrates how to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
curl -v -X POST
"https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
-H "Content-type: application/x-www-form-urlencoded" \
-H "Content-Length: 0" \
-H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

C# sample

This C# class illustrates how to get an access token. Pass your Speech Service subscription key when you instantiate the class. If your subscription isn't in the West US region, change the value of `FetchTokenUri` to match the region for your subscription.

```

public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
    private string subscriptionKey;
    private string token;

    public Authentication(string subscriptionKey)
    {
        this.subscriptionKey = subscriptionKey;
        this.token = FetchTokenAsync(FetchTokenUri, subscriptionKey).Result;
    }

    public string GetAccessToken()
    {
        return this.token;
    }

    private async Task<string> FetchTokenAsync(string fetchUri, string subscriptionKey)
    {
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
            UriBuilder uriBuilder = new UriBuilder(fetchUri);

            var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
            Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
            return await result.Content.ReadAsStringAsync();
        }
    }
}

```

Python sample

```

# Request module must be installed.
# Run pip install requests if necessary.
import requests

subscription_key = 'REPLACE_WITH_YOUR_KEY'

def get_token(subscription_key):
    fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key
    }
    response = requests.post(fetch_token_url, headers=headers)
    access_token = str(response.text)
    print(access_token)

```

How to use an access token

The access token should be sent to the service as the `Authorization: Bearer <TOKEN>` header. Each access token is valid for 10 minutes. You can get a new token at any time, however, to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the text-to-speech REST API:

```

POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.stt.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

// Message body here...

```

Regions and endpoints

These regions are supported for speech-to-text transcription using the REST API. Make sure that you select the endpoint that matches your subscription region.

REGION	ENDPOINT
Australia East	https://australiaeast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
Canada Central	https://canadacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
Central US	https://centralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1

REGION	ENDPOINT
East Asia	https://eastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
East US	https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
East US 2	https://eastus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
France Central	https://francecentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
India Central	https://centralindia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
Japan East	https://japaneast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
Korea Central	https://koreacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
North Central US	https://northcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
North Europe	https://northeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
South Central US	https://southcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
Southeast Asia	https://southeastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
UK South	https://uksouth.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
West Europe	https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
West US	https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1
West US 2	https://westus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1

NOTE

The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is: <https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US>.

Query parameters

These parameters may be included in the query string of the REST request.

PARAMETER	DESCRIPTION	REQUIRED / OPTIONAL
<code>language</code>	Identifies the spoken language that is being recognized. See Supported languages .	Required
<code>format</code>	Specifies the result format. Accepted values are <code>simple</code> and <code>detailed</code> . Simple results include <code>RecognitionStatus</code> , <code>DisplayText</code> , <code>Offset</code> , and <code>Duration</code> . Detailed responses include multiple results with confidence values and four different representations. The default setting is <code>simple</code> .	Optional
<code>profanity</code>	Specifies how to handle profanity in recognition results. Accepted values are <code>masked</code> , which replaces profanity with asterisks, <code>removed</code> , which removes all profanity from the result, or <code>raw</code> , which includes the profanity in the result. The default setting is <code>masked</code> .	Optional

Request headers

This table lists required and optional headers for speech-to-text requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
<code>Ocp-Apim-Subscription-Key</code>	Your Speech service subscription key.	Either this header or <code>Authorization</code> is required.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
Authorization	An authorization token preceded by the word <code>Bearer</code> . For more information, see Authentication .	Either this header or <code>Ocp-Apim-Subscription-Key</code> is required.
Content-type	Describes the format and codec of the provided audio data. Accepted values are <code>audio/wav; codecs=audio/pcm; samplerate=16000</code> and <code>audio/ogg; codecs=opus</code> .	Required
Transfer-Encoding	Specifies that chunked audio data is being sent, rather than a single file. Only use this header if chunking audio data.	Optional
Expect	If using chunked transfer, send <code>Expect: 100-continue</code> . The Speech service acknowledges the initial request and awaits additional data.	Required if sending chunked audio data.
Accept	If provided, it must be <code>application/json</code> . The Speech service provides results in JSON. Some request frameworks provide an incompatible default value. It is good practice to always include <code>Accept</code> .	Optional, but recommended.

Audio formats

Audio is sent in the body of the HTTP `POST` request. It must be in one of the formats in this table:

FORMAT	CODEC	BITRATE	SAMPLE RATE
WAV	PCM	16-bit	16 kHz, mono
OGG	OPUS	16-bit	16 kHz, mono

NOTE

The above formats are supported through REST API and WebSocket in the Speech service. The [Speech SDK](#) currently only supports the WAV format with PCM codec.

Sample request

The sample below includes the hostname and required headers. It's important to note that the service also expects audio data, which is not included in this sample. As mentioned earlier, chunking is recommended, however, not required.

```
POST speech/recognition/conversation/cognitiveservices/v1?language=en-US&format=detailed HTTP/1.1
Accept: application/json;text/xml
Content-Type: audio/wav; codecs=audio/pcm; samplerate=16000
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.stt.speech.microsoft.com
Transfer-Encoding: chunked
Expect: 100-continue
```

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
100	Continue	The initial request has been accepted. Proceed with sending the rest of the data. (Used with chunked transfer.)
200	OK	The request was successful; the response body is a JSON object.
400	Bad request	Language code not provided, not a supported language, invalid audio file, etc.
401	Unauthorized	Subscription key or authorization token is invalid in the specified region, or invalid endpoint.
403	Forbidden	Missing subscription key or authorization token.

Chunked transfer

Chunked transfer (`Transfer-Encoding: chunked`) can help reduce recognition latency. It allows the Speech service to begin processing the audio file while it is transmitted. The REST API does not provide partial or interim results.

This code sample shows how to send audio in chunks. Only the first chunk should contain the audio file's header. `request` is an `HTTPWebRequest` object connected to the appropriate REST endpoint. `audioFile` is the path to an audio file on disk.

```
HttpWebRequest request = null;
request = (HttpWebRequest)HttpWebRequest.Create(requestUri);
request.SendChunked = true;
request.Accept = @"application/json;text/xml";
request.Method = "POST";
request.ProtocolVersion = HttpVersion.Version11;
request.Host = host;
request.ContentType = @"audio/wav; codecs=audio/pcm; samplerate=16000";
request.Headers["Ocp-Apim-Subscription-Key"] = args[1];
request.AllowReadStreamBuffering = false;

using (fs = new FileStream(audioFile, FileMode.Open, FileAccess.Read))
{
    /*
     * Open a request stream and write 1024 byte chunks in the stream one at a time.
     */
    byte[] buffer = null;
    int bytesRead = 0;
    using (Stream requestStream = request.GetRequestStream())
    {
        /*
         * Read 1024 raw bytes from the input audio file.
         */
        buffer = new Byte[checked((uint)Math.Min(1024, (int)fs.Length))];
        while ((bytesRead = fs.Read(buffer, 0, buffer.Length)) != 0)
        {
            requestStream.Write(buffer, 0, bytesRead);
        }

        // Flush
        requestStream.Flush();
    }
}
```

Response parameters

Results are provided as JSON. The `simple` format includes these top-level fields.

PARAMETER	DESCRIPTION
<code>RecognitionStatus</code>	Status, such as <code>Success</code> for successful recognition. See next table.
<code>DisplayText</code>	The recognized text after capitalization, punctuation, inverse text normalization (conversion of spoken text to shorter forms, such as 200 for "two hundred" or "Dr. Smith" for "doctor smith"), and profanity masking. Present only on success.
<code>Offset</code>	The time (in 100-nanosecond units) at which the recognized speech begins in the audio stream.
<code>Duration</code>	The duration (in 100-nanosecond units) of the recognized speech in the audio stream.

The `RecognitionStatus` field may contain these values:

STATUS	DESCRIPTION
<code>Success</code>	The recognition was successful and the <code>DisplayText</code> field is present.
<code>NoMatch</code>	Speech was detected in the audio stream, but no words from the target language were matched. Usually means the recognition language is a different language from the one the user is speaking.
<code>InitialSilenceTimeout</code>	The start of the audio stream contained only silence, and the service timed out waiting for speech.
<code>BabbleTimeout</code>	The start of the audio stream contained only noise, and the service timed out waiting for speech.

STATUS	DESCRIPTION
Error	The recognition service encountered an internal error and could not continue. Try again if possible.

NOTE

If the audio consists only of profanity, and the `profanity` query parameter is set to `remove`, the service does not return a speech result.

The `detailed` format includes the same data as the `simple` format, along with `NBest`, a list of alternative interpretations of the same recognition result. These results are ranked from most likely to least likely. The first entry is the same as the main recognition result. When using the `detailed` format, `DisplayText` is provided as `Display` for each result in the `NBest` list.

Each object in the `NBest` list includes:

PARAMETER	DESCRIPTION
<code>Confidence</code>	The confidence score of the entry from 0.0 (no confidence) to 1.0 (full confidence)
<code>Lexical</code>	The lexical form of the recognized text: the actual words recognized.
<code>ITN</code>	The inverse-text-normalized ("canonical") form of the recognized text, with phone numbers, numbers, abbreviations ("doctor smith" to "dr smith"), and other transformations applied.
<code>MaskedITN</code>	The ITN form with profanity masking applied, if requested.
<code>Display</code>	The display form of the recognized text, with punctuation and capitalization added. This parameter is the same as <code>DisplayText</code> provided when format is set to <code>simple</code> .

Sample responses

A typical response for `simple` recognition:

```
{
  "RecognitionStatus": "Success",
  "DisplayText": "Remind me to buy 5 pencils.",
  "Offset": "1236645672289",
  "Duration": "1236645672289"
}
```

A typical response for `detailed` recognition:

```
{
  "RecognitionStatus": "Success",
  "Offset": "1236645672289",
  "Duration": "1236645672289",
  "NBest": [
    {
      "Confidence" : "0.87",
      "Lexical" : "remind me to buy five pencils",
      "ITN" : "remind me to buy 5 pencils",
      "MaskedITN" : "remind me to buy 5 pencils",
      "Display" : "Remind me to buy 5 pencils.",
    },
    {
      "Confidence" : "0.54",
      "Lexical" : "rewind me to buy five pencils",
      "ITN" : "rewind me to buy 5 pencils",
      "MaskedITN" : "rewind me to buy 5 pencils",
      "Display" : "Rewind me to buy 5 pencils."
    }
  ]
}
```

Next steps

- [Get your Speech trial subscription](#)
- [Customize acoustic models](#)
- [Customize language models](#)

How to use batch transcription

12/20/2019 • 9 minutes to read • [Edit Online](#)

Batch transcription is ideal for transcribing a large amount of audio in storage. By using the dedicated REST API, you can point to audio files with a shared access signature (SAS) URI and asynchronously receive transcription results.

The API offers asynchronous speech-to-text transcription and other features. You can use REST API to expose methods to:

- Create a batch processing requests
- Query the status
- Download transcription results
- Delete transcription information from the service

The detailed API is available as a [Swagger document](#), under the heading `Custom Speech transcriptions`.

Batch transcription jobs are scheduled on a best effort basis. Currently there is no estimate for when a job will change into the running state. Under normal system load, it should happen within minutes. Once in the running state, the actual transcription is processed faster than the audio real time.

Next to the easy-to-use API, you don't need to deploy custom endpoints, and you don't have any concurrency requirements to observe.

Prerequisites

Subscription Key

As with all features of the Speech service, you create a subscription key from the [Azure portal](#) by following our [Get started guide](#).

NOTE

A standard subscription (S0) for Speech service is required to use batch transcription. Free subscription keys (F0) will not work. For more information, see [pricing and limits](#).

Custom models

If you plan to customize acoustic or language models, follow the steps in [Customize acoustic models](#) and [Design customization language models](#). To use the created models in batch transcription, you need their model IDs. You can retrieve the model ID when you inspect the details of the model. A deployed custom endpoint is not needed for the batch transcription service.

The Batch Transcription API

Supported formats

The Batch Transcription API supports the following formats:

Format	Codec	Bitrate	Sample rate
WAV	PCM	16-bit	8 kHz or 16 kHz, mono or stereo

FORMAT	CODEC	BITRATE	SAMPLE RATE
MP3	PCM	16-bit	8 kHz or 16 kHz, mono or stereo
OGG	OPUS	16-bit	8 kHz or 16 kHz, mono or stereo

For stereo audio streams, the left and right channels are split during the transcription. For each channel, a JSON result file is being created. The timestamps generated per utterance enable the developer to create an ordered final transcript.

Configuration

Configuration parameters are provided as JSON:

```
{
  "recordingsUrl": "<URL to the Azure blob to transcribe>",
  "models": [{"Id": "<optional acoustic model ID>"}, {"Id": "<optional language model ID>"}],
  "locale": "<locale to use, for example en-US>",
  "name": "<user defined name of the transcription batch>",
  "description": "<optional description of the transcription>",
  "properties": {
    "ProfanityFilterMode": "None | Removed | Tags | Masked",
    "PunctuationMode": "None | Dictated | Automatic | DictatedAndAutomatic",
    "AddWordLevelTimestamps" : "True | False",
    "AddSentiment" : "True | False",
    "AddDiarization" : "True | False",
    "TranscriptionResultsContainerUrl" : "<SAS to Azure container to store results into (write permission required)>"
  }
}
```

Configuration properties

Use these optional properties to configure transcription:

PARAMETER	DESCRIPTION
ProfanityFilterMode	Specifies how to handle profanity in recognition results <div style="display: flex; justify-content: space-around;"> Masked - Default. Replaces profanity with asterisks None - Disables profanity filtering Removed - Removes all profanity from the result Tags - Adds profanity tags </div>
PunctuationMode	Specifies to handle punctuation in recognition results <div style="display: flex; justify-content: space-around;"> Automatic - The service inserts punctuation Dictated - Dictated (spoken) punctuation DictatedAndAutomatic - Default. Dictated and automatic punctuation None - Disables punctuation </div>
AddWordLevelTimestamps	Specifies if word level timestamps should be added to the output

PARAMETER	DESCRIPTION
	<p><code>True</code> - Enables word level timestamps <code>False</code> - Default. Disable word level timestamps</p>
<code>AddSentiment</code>	Specifies if sentiment analysis is added to the utterance
	<p><code>True</code> - Enables sentiment per utterance <code>False</code> - Default. Disable sentiment</p>
<code>AddDiarization</code>	Specifies if diarization analysis is carried out. If <code>true</code> , the input is expected to be mono channel audio containing a maximum of two voices. <code>AddWordLevelTimestamps</code> needs to be set to <code>true</code>
	<p><code>True</code> - Enables diarization <code>False</code> - Default. Disable diarization</p>
<code>TranscriptionResultsContainerUrl</code>	Optional SAS token to a writeable container in Azure. The result will be stored in this container

Storage

Batch transcription supports [Azure Blob storage](#) for reading audio and writing transcriptions to storage.

The batch transcription result

For mono input audio, one transcription result file is being created. For stereo input audio, two transcription result files are being created. Each has this structure:

```

{
  "AudioFileResults": [
    {
      "AudioFileName": "Channel.0.wav | Channel.1.wav"           'maximum of 2 channels supported'
      "AudioFileUrl": null                                     'always null'
      "AudioLengthInSeconds": number                         'Real number. Two decimal places'
      "CombinedResults": [
        {
          "ChannelNumber": null                                'always null'
          "Lexical": string
          "ITN": string
          "MaskedITN": string
          "Display": string
        }
      ]
    }
  ]
  SegmentResults: [                                         'for each individual segment'
    {
      "RecognitionStatus": Success | Failure
      "ChannelNumber": null
      "SpeakerId": null | "1 | 2"                           'null if no diarization
                                                               or stereo input file, the
                                                               speakerId as a string if
                                                               diarization requested for
                                                               mono audio file'
      "Offset": number                                     'time in milliseconds'
      "Duration": number                                 'time in milliseconds'
      "OffsetInSeconds": : number                        'Real number. Two decimal places'
      "DurationInSeconds": : number                      'Real number. Two decimal places'
      "NBest": [
        {
          "Confidence": number                            'between 0 and 1'
          "Lexical": string
          "ITN": string
          "MaskedITN": string
          "Display": string
          "Sentiment": {
            {
              "Negative": number                         'this is omitted if sentiment is
                                                               not requested'
              "Neutral": number                         'between 0 and 1'
              "Positive": number                        'between 0 and 1'
            }
          }
          "Words": [
            {
              "Word": string
              "Offset": number                         'time in milliseconds'
              "Duration": number                       'time in milliseconds'
              "OffsetInSeconds": number                 'Real number. Two decimal places'
              "DurationInSeconds": number               'Real number. Two decimal places'
            }
          ]
        }
      ]
    }
  ]
}

```

The result contains these forms:

FORM	CONTENT
Lexical	The actual words recognized.

FORM	CONTENT
ITN	Inverse-text-normalized form of the recognized text. Abbreviations ("doctor smith" to "dr smith"), phone numbers, and other transformations are applied.
MaskedITN	The ITN form with profanity masking applied.
Display	The display form of the recognized text. This includes added punctuation and capitalization.

Speaker separation (Diarization)

Diarization is the process of separating speakers in a piece of audio. Our Batch pipeline supports diarization and is capable of recognizing two speakers on mono channel recordings. The feature is not available on stereo recordings.

All transcription output contains a `SpeakerId`. If diarization is not used, it will show `"SpeakerId": null` in the JSON output. For diarization we support two voices, so the speakers will be identified as `"1"` or `"2"`.

To request diarization, you simply have to add the relevant parameter in the HTTP request as shown below.

```
{
  "recordingsUrl": "<URL to the Azure blob to transcribe>",
  "models": [{"Id": "<optional acoustic model ID>"}, {"Id": "<optional language model ID>"}],
  "locale": "<locale to us, for example en-US>",
  "name": "<user defined name of the transcription batch>",
  "description": "<optional description of the transcription>",
  "properties": {
    "AddWordLevelTimestamps" : "True",
    "AddDiarization" : "True"
  }
}
```

Word-level timestamps would also have to be 'turned on' as the parameters in the above request indicate.

Sentiment analysis

The sentiment feature estimates the sentiment expressed in the audio. The sentiment is expressed by a value between 0 and 1 for `Negative`, `Neutral`, and `Positive` sentiment. For example, sentiment analysis can be used in call center scenarios:

- Get insight on customer satisfaction
- Get insight on the performance of the agents (team taking the calls)
- Find the exact point in time when a call took a turn in a negative direction
- What went well when turning a negative call into a positive direction
- Identify what customers like and what they dislike about a product or a service

Sentiment is scored per audio segment based on the lexical form. The entire text within that audio segment is used to calculate sentiment. No aggregate sentiment is being calculated for the entire transcription.

A JSON output sample looks like below:

```
{  
  "AudioFileResults": [  
    {  
      "AudioFileName": "Channel.0.wav",  
      "AudioFileUrl": null,  
      "SegmentResults": [  
        {  
          "RecognitionStatus": "Success",  
          "ChannelNumber": null,  
          "Offset": 400000,  
          "Duration": 13300000,  
          "NBest": [  
            {  
              "Confidence": 0.976174,  
              "Lexical": "what's the weather like",  
              "ITN": "what's the weather like",  
              "MaskedITN": "what's the weather like",  
              "Display": "What's the weather like?",  
              "Words": null,  
              "Sentiment": {  
                "Negative": 0.206194,  
                "Neutral": 0.793785,  
                "Positive": 0.0  
              }  
            }  
          ]  
        ]  
      ]  
    ]  
  ]  
}
```

Best practices

The transcription service can handle large number of submitted transcriptions. You can query the status of your transcriptions through a `GET` on the [transcriptions method](#). Keep the information returned to a reasonable size by specifying the `take` parameter (a few hundred). [Delete transcriptions](#) regularly from the service once you retrieved the results. This will guarantee quick replies from the transcription management calls.

Sample code

Complete samples are available in the [GitHub sample repository](#) inside the `samples/batch` subdirectory.

You have to customize the sample code with your subscription information, the service region, the SAS URI pointing to the audio file to transcribe, and model IDs in case you want to use a custom acoustic or language model.

```
// Replace with your subscription key
private const string SubscriptionKey = "YourSubscriptionKey";

// Update with your service region
private const string Region = "YourServiceRegion";
private const int Port = 443;

// recordings and locale
private const string Locale = "en-US";
private const string RecordingsBlobUri = "<SAS URI pointing to an audio file stored in Azure Blob Storage>";

// For usage of baseline models, no acoustic and language model needs to be specified.
private static Guid[] modelList = new Guid[0];

// For use of specific acoustic and language models:
// - comment the previous line
// - uncomment the next lines to create an array containing the guids of your required model(s)
// private static Guid AdaptedAcousticId = new Guid("<id of the custom acoustic model>");
// private static Guid AdaptedLanguageId = new Guid("<id of the custom language model>");
// private static Guid[] modelList = new[] { AdaptedAcousticId, AdaptedLanguageId };

//name and description
private const string Name = "Simple transcription";
private const string Description = "Simple transcription description";
```

The sample code will set up the client and submit the transcription request. It will then poll for status information and print details about the transcription progress.

```

// get all transcriptions for the user
transcriptions = await client.GetTranscriptionsAsync().ConfigureAwait(false);

completed = 0; running = 0; notStarted = 0;
// for each transcription in the list we check the status
foreach (var transcription in transcriptions)
{
    switch (transcription.Status)
    {
        case "Failed":
        case "Succeeded":
            // we check to see if it was one of the transcriptions we created from this client.
            if (!createdTranscriptions.Contains(transcription.Id))
            {
                // not created from here, continue
                continue;
            }
            completed++;

            // if the transcription was successful, check the results
            if (transcription.Status == "Succeeded")
            {
                var resultsUri0 = transcription.ResultsUrls["channel_0"];

                WebClient webClient = new WebClient();

                var filename = Path.GetTempFileName();
                webClient.DownloadFile(resultsUri0, filename);
                var results0 = File.ReadAllText(filename);
                var resultObject0 = JsonConvert.DeserializeObject<RootObject>(results0);
                Console.WriteLine(resultObject0);

                Console.WriteLine("Transcription succeeded. Results: ");
                Console.WriteLine(resultObject0);
            }
            else
            {
                Console.WriteLine("Transcription failed. Status: {0}", transcription.StatusMessage);
            }
            break;

        case "Running":
            running++;
            break;

        case "NotStarted":
            notStarted++;
            break;
    }
}

```

For full details about the preceding calls, see our [Swagger document](#). For the full sample shown here, go to [GitHub](#) in the `samples/batch` subdirectory.

Take note of the asynchronous setup for posting audio and receiving transcription status. The client that you create is a .NET HTTP client. There's a `PostTranscriptions` method for sending the audio file details and a `GetTranscriptions` method for receiving the results. `PostTranscriptions` returns a handle, and `GetTranscriptions` uses it to create a handle to get the transcription status.

The current sample code doesn't specify a custom model. The service uses the baseline models for transcribing the file or files. To specify the models, you can pass on the same method as the model IDs for the acoustic and the language model.

NOTE

For baseline transcriptions, you don't need to declare the ID for the baseline models. If you only specify a language model ID (and no acoustic model ID), a matching acoustic model is automatically selected. If you only specify an acoustic model ID, a matching language model is automatically selected.

Download the sample

You can find the sample in the `samples/batch` directory in the [GitHub sample repository](#).

Next steps

- [Get your Speech trial subscription](#)

Speech to Text frequently asked questions

12/4/2019 • 8 minutes to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, check out [other support options](#).

General

Q: What is the difference between a baseline model and a custom Speech to Text model?

A: A baseline model has been trained by using Microsoft-owned data and is already deployed in the cloud. You can use a custom model to adapt a model to better fit a specific environment that has specific ambient noise or language. Factory floors, cars, or noisy streets would require an adapted acoustic model. Topics like biology, physics, radiology, product names, and custom acronyms would require an adapted language model.

Q: Where do I start if I want to use a baseline model?

A: First, get a [subscription key](#). If you want to make REST calls to the predeployed baseline models, see the [REST APIs](#). If you want to use WebSockets, [download the SDK](#).

Q: Do I always need to build a custom speech model?

A: No. If your application uses generic, day-to-day language, you don't need to customize a model. If your application is used in an environment where there's little or no background noise, you don't need to customize a model.

You can deploy baseline and customized models in the portal and then run accuracy tests against them. You can use this feature to measure the accuracy of a baseline model versus a custom model.

Q: How will I know when processing for my dataset or model is complete?

A: Currently, the status of the model or dataset in the table is the only way to know. When the processing is complete, the status is **Succeeded**.

Q: Can I create more than one model?

A: There's no limit on the number of models you can have in your collection.

Q: I realized I made a mistake. How do I cancel my data import or model creation that's in progress?

A: Currently, you can't roll back an acoustic or language adaptation process. You can delete imported data and models when they're in a terminal state.

Q: What's the difference between the Search and Dictation model and the Conversational model?

A: You can choose from more than one baseline model in the Speech service. The Conversational model is useful for recognizing speech that is spoken in a conversational style. This model is ideal for transcribing phone calls. The Search and Dictation model is ideal for voice-triggered apps. The Universal model is a new model that aims to address both scenarios. The Universal model is currently at or above the quality level of the Conversational model in most locales.

Q: Can I update my existing model (model stacking)?

A: You can't update an existing model. As a solution, combine the old dataset with the new dataset and readapt.

The old dataset and the new dataset must be combined in a single .zip file (for acoustic data) or in a .txt file (for language data). When adaptation is finished, the new, updated model needs to be redeployed to obtain a new

endpoint

Q: When a new version of a baseline is available, is my deployment automatically updated?

A: Deployments will NOT be automatically updated.

If you have adapted and deployed a model with baseline V1.0, that deployment will remain as is. Customers can decommission the deployed model, readapt using the newer version of the baseline and redeploy.

Q: Can I download my model and run it locally?

A: Models can't be downloaded and executed locally.

Q: Are my requests logged?

A: You have a choice when you create a deployment to switch off tracing. At that point, no audio or transcriptions will be logged. Otherwise, requests are typically logged in Azure in secure storage.

Q: Are my requests throttled?

A: The REST API limits requests to 25 per 5 seconds. Details can be found in our pages for [Speech to text](#).

Q: How I am charged for dual channel audio?

A: If you submit each channel separately (each channel in its own file), you will be charged per duration of file. If you submit a single file with each channel multiplexed together, then you will be charged for the duration of the single file.

IMPORTANT

If you have further privacy concerns that prohibit you from using the custom Speech service, contact one of the support channels.

Increasing concurrency

Q: What if I need higher concurrency for my deployed model than what is offered in the portal?

A: You can scale up your model in increments of 20 concurrent requests.

With the required information, create a support request in the [Azure support portal](#). Do not post the information on any of the public channels (GitHub, Stackoverflow, ...) mentioned on the [support page](#).

To increase concurrency for a **custom model**, we need the following information:

- The region where the model is deployed,
- the endpoint ID of the deployed model:
 - Got to the [Custom Speech Portal](#),
 - sign in (if necessary),
 - select your project and deployment,
 - select the endpoint you need the concurrency increase for,
 - copy the `Endpoint ID`.

To increase concurrency for a **base model**, we need the following information:

- The region of your service,
- and either
- an access token for your subscription (see [here](#)),

or

- the Resource ID for your subscription:
 - Go to the [Azure portal](#),
 - select `Cognitive Services` in the search box,
 - from the displayed services pick the speech service you want the concurrency increased for,
 - display the `Properties` for this service,
 - copy the complete `Resource ID`.

Importing data

Q: What is the limit on the size of a dataset, and why is it the limit?

A: The current limit for a dataset is 2 GB. The limit is due to the restriction on the size of a file for HTTP upload.

Q: Can I zip my text files so I can upload a larger text file?

A: No. Currently, only uncompressed text files are allowed.

Q: The data report says there were failed utterances. What is the issue?

A: Failing to upload 100 percent of the utterances in a file is not a problem. If the vast majority of the utterances in an acoustic or language dataset (for example, more than 95 percent) are successfully imported, the dataset can be usable. However, we recommend that you try to understand why the utterances failed and fix the problems. Most common problems, such as formatting errors, are easy to fix.

Creating an acoustic model

Q: How much acoustic data do I need?

A: We recommend starting with between 30 minutes and one hour of acoustic data.

Q: What data should I collect?

A: Collect data that's as close to the application scenario and use case as possible. The data collection should match the target application and users in terms of device or devices, environments, and types of speakers. In general, you should collect data from as broad a range of speakers as possible.

Q: How should I collect acoustic data?

A: You can create a standalone data collection application or use off-the-shelf audio recording software. You can also create a version of your application that logs the audio data and then uses the data.

Q: Do I need to transcribe adaptation data myself?

A: Yes! You can transcribe it yourself or use a professional transcription service. Some users prefer professional transcribers and others use crowdsourcing or do the transcriptions themselves.

Accuracy testing

Q: Can I perform offline testing of my custom acoustic model by using a custom language model?

A: Yes, just select the custom language model in the drop-down menu when you set up the offline test.

Q: Can I perform offline testing of my custom language model by using a custom acoustic model?

A: Yes, just select the custom acoustic model in the drop-down menu when you set up the offline test.

Q: What is word error rate (WER) and how is it computed?

A: WER is the evaluation metric for speech recognition. WER is counted as the total number of errors, which includes insertions, deletions, and substitutions, divided by the total number of words in the reference transcription. For more information, see [word error rate](#).

Q: How do I determine whether the results of an accuracy test are good?

A: The results show a comparison between the baseline model and the model you customized. You should aim to beat the baseline model to make customization worthwhile.

Q: How do I determine the WER of a base model so I can see if there was an improvement?

A: The offline test results show the baseline accuracy of the custom model and the improvement over baseline.

Creating a language model

Q: How much text data do I need to upload?

A: It depends on how different the vocabulary and phrases used in your application are from the starting language models. For all new words, it's useful to provide as many examples as possible of the usage of those words. For common phrases that are used in your application, including phrases in the language data is also useful because it tells the system to also listen for these terms. It's common to have at least 100, and typically several hundred or more utterances in the language dataset. Also, if some types of queries are expected to be more common than others, you can insert multiple copies of the common queries in the dataset.

Q: Can I just upload a list of words?

A: Uploading a list of words will add the words to the vocabulary, but it won't teach the system how the words are typically used. By providing full or partial utterances (sentences or phrases of things that users are likely to say), the language model can learn the new words and how they are used. The custom language model is good not only for adding new words to the system, but also for adjusting the likelihood of known words for your application. Providing full utterances helps the system learn better.

Tenant Model (Custom Speech with Office 365 data)

Q: What information is included in the Tenant Model, and how is it created?

A: A Tenant Model is built using [public group](#) emails and documents that can be seen by anyone in your organization.

Q: What speech experiences are improved by the Tenant Model?

A: When the Tenant Model is enabled, created and published, it is used to improve recognition for any enterprise applications built using the Speech service; that also pass a user AAD token indicating membership to the enterprise.

The speech experiences built into Office 365, such as Dictation and PowerPoint Captioning, aren't changed when you create a Tenant Model for your Speech service applications.

Next steps

- [Troubleshooting](#)
- [Release notes](#)

What is text-to-speech?

12/16/2019 • 3 minutes to read • [Edit Online](#)

Text-to-speech from the Speech service enables your applications, tools, or devices to convert text into human-like synthesized speech. Choose from standard and neural voices, or create a custom voice unique to your product or brand. 75+ standard voices are available in more than 45 languages and locales, and 5 neural voices are available in a select number of languages and locales. For a full list of supported voices, languages, and locales, see [supported languages](#).

NOTE

Bing Speech was decommissioned on October 15, 2019. If your applications, tools, or products are using the Bing Speech APIs or Custom Speech, we've created guides to help you migrate to the Speech service.

- [Migrate from Bing Speech to the Speech service](#)

Core features

- Speech synthesis - Use the [Speech SDK](#) or [REST API](#) to convert text-to-speech using standard, neural, or custom voices.
- Asynchronous synthesis of long audio - Use the [Long Audio API](#) to asynchronously synthesize text-to-speech files longer than 10 minutes (for example audio books or lectures). Unlike synthesis performed using the Speech SDK or speech-to-text REST API, responses aren't returned in real time. The expectation is that requests are sent asynchronously, responses are polled for, and that the synthesized audio is downloaded when made available from the service. Only neural voices are supported.
- Standard voices - Created using Statistical Parametric Synthesis and/or Concatenation Synthesis techniques. These voices are highly intelligible and sound natural. You can easily enable your applications to speak in more than 45 languages, with a wide range of voice options. These voices provide high pronunciation accuracy, including support for abbreviations, acronym expansions, date/time interpretations, polyphones, and more. For a full list of standard voices, see [supported languages](#).
- Neural voices - Deep neural networks are used to overcome the limits of traditional speech synthesis with regards to stress and intonation in spoken language. Prosody prediction and voice synthesis are performed simultaneously, which results in more fluid and natural-sounding outputs. Neural voices can be used to make interactions with chatbots and voice assistants more natural and engaging, convert digital texts such as e-books into audiobooks, and enhance in-car navigation systems. With the human-like natural prosody and clear articulation of words, neural voices significantly reduce listening fatigue when you interact with AI systems. For a full list of neural voices, see [supported languages](#).
- Speech Synthesis Markup Language (SSML) - An XML-based markup language used to customize speech-to-text outputs. With SSML, you can adjust pitch, add pauses, improve pronunciation, speed up or slow down speaking rate, increase or decrease volume, and attribute multiple voices to a single document. See [SSML](#).

Get started

The text-to-speech service is available via the [Speech SDK](#). There are several common scenarios available as quickstarts, in various languages and platforms:

- [Synthesize speech into an audio file](#)
- [Synthesize speech to a speaker](#)
- [Asynchronously synthesize long-form audio](#)

If you prefer, the text-to-speech service is accessible via [REST](#).

Sample code

Sample code for text-to-speech is available on GitHub. These samples cover text-to-speech conversion in most popular programming languages.

- [Text-to-speech samples \(SDK\)](#)
- [Text-to-speech samples \(REST\)](#)

Customization

In addition to standard and neural voices, you can create and fine-tune custom voices unique to your product or brand. All it takes to get started are a handful of audio files and the associated transcriptions. For more information, see [Get started with Custom Voice](#)

Pricing note

When using the text-to-speech service, you are billed for each character that is converted to speech, including punctuation. While the SSML document itself is not billable, optional elements that are used to adjust how the text is converted to speech, like phonemes and pitch, are counted as billable characters. Here's a list of what's billable:

- Text passed to the text-to-speech service in the SSML body of the request
- All markup within the text field of the request body in the SSML format, except for `<speak>` and `<voice>` tags
- Letters, punctuation, spaces, tabs, markup, and all white-space characters
- Every code point defined in Unicode

For detailed information, see [Pricing](#).

IMPORTANT

Each Chinese, Japanese, and Korean language character is counted as two characters for billing.

Reference docs

- [Speech SDK](#)
- [REST API: Text-to-speech](#)

Next steps

- [Get a free Speech service subscription](#)
- [Get the Speech SDK](#)

Quickstart: Synthesize speech into an audio file

12/4/2019 • 12 minutes to read • [Edit Online](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech in an audio file. After satisfying a few prerequisites, synthesizing speech into a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an Audio Configuration object that specifies the .WAV file name.
- Create a `SpeechSynthesizer` object using the configuration objects from above.
- Using the `SpeechSynthesizer` object, convert your text into synthesized speech, saving it into the audio file specified.
- Inspect the `SpeechSynthesizer` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Open your project in Visual Studio

The first step is to make sure that you have your project open in Visual Studio.

1. Launch Visual Studio 2019.
2. Load your project and open `Program.cs`.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project. Make note that you've created an async method called `SynthesisToFileAsync()`.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task SynthesisToAudioFileAsync()
        {

        }

        static void Main()
        {
            SynthesisToAudioFileAsync().Wait();
        }
    }
}
```

Create a Speech configuration

Before you can initialize a `SpeechSynthesizer` object, you need to create a configuration that uses your subscription key and subscription region. Insert this code in the `SynthesisToAudioFileAsync()` method.

```
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

Create an Audio configuration

Now, you need to create an `AudioConfig` object that points to your audio file. This object is created inside of a `using` statement to ensure the proper release of unmanaged resources. Insert this code in the `SynthesisToAudioFileAsync()` method, right below your Speech configuration.

```
var fileName = "helloworld.wav";
using (var fileOutput = AudioConfig.FromWavFileOutput(fileName))
{}
```

Initialize a SpeechSynthesizer

Now, let's create the `SpeechSynthesizer` object using the `SpeechConfig` and `AudioConfig` objects created earlier. This object is also created inside of a `using` statement to ensure the proper release of unmanaged resources. Insert this code in the `SynthesisToAudioFileAsync()` method, inside the `using` statement that wraps your `AudioConfig` object.

```
using (var synthesizer = new SpeechSynthesizer(config, fileOutput))
{}
```

Synthesize text using SpeakTextAsync

From the `SpeechSynthesizer` object, you're going to call the `SpeakTextAsync()` method. This method sends your text to the Speech service which converts it to audio. The `SpeechSynthesizer` will use the default voice if

`config.VoiceName` isn't explicitly specified.

Inside the using statement, add this code:

```
var text = "Hello world!";
var result = await synthesizer.SpeakTextAsync(text);
```

Check for errors

When the synthesis result is returned by the Speech service, you should check to make sure your text was successfully synthesized.

Inside the using statement, below `SpeakTextAsync()`, add this code:

```
if (result.Reason == ResultReason.SynthesizingAudioCompleted)
{
    Console.WriteLine($"Speech synthesized to [{fileName}] for text [{text}]");
}
else if (result.Reason == ResultReason.Canceled)
{
    var cancellation = SpeechSynthesisCancellationDetails.FromResult(result);
    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

    if (cancellation.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails=[{cancellation.ErrorDetails}]");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }
}
```

Check your code

At this point, your code should look like this:

```

// Copyright (c) Microsoft. All rights reserved.
// Licensed under the MIT license. See LICENSE.md file in the project root for full license information.
//

using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task SynthesisToAudioFileAsync()
        {
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            var fileName = "helloworld.wav";
            using (var fileOutput = AudioConfig.FromWavFileOutput(fileName))
            {
                using (var synthesizer = new SpeechSynthesizer(config, fileOutput))
                {
                    var text = "Hello world!";
                    var result = await synthesizer.SpeakTextAsync(text);

                    if (result.Reason == ResultReason.SynthesizingAudioCompleted)
                    {
                        Console.WriteLine($"Speech synthesized to [{fileName}] for text [{text}]");
                    }
                    else if (result.Reason == ResultReason.Canceled)
                    {
                        var cancellation = SpeechSynthesisCancellationDetails.FromResult(result);
                        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

                        if (cancellation.Reason == CancellationReason.Error)
                        {
                            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                            Console.WriteLine($"CANCELED: ErrorDetails=[{cancellation.ErrorDetails}]");
                            Console.WriteLine($"CANCELED: Did you update the subscription info?");
                        }
                    }
                }
            }

            static void Main()
            {
                SynthesisToAudioFileAsync().Wait();
            }
        }
    }
}

```

Build and run your app

Now you're ready to build your app and test our speech synthesis using the Speech service.

- Compile the code** - From the menu bar of Visual Studio, choose **Build > Build Solution**.
- Start your app** - From the menu bar, choose **Debug > Start Debugging** or press **F5**.
- Start synthesis** - Your text is converted to speech, and saved in the audio data specified.

Speech synthesized to [helloworld.wav] for text [Hello world!]

Next steps

[Explore C# samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech in an audio file. After satisfying a few prerequisites, synthesizing speech into a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an Audio Configuration object that specifies the .WAV file name.
- Create a `SpeechSynthesizer` object using the configuration objects from above.
- Using the `SpeechSynthesizer` object, convert your text into synthesized speech, saving it into the audio file specified.
- Inspect the `SpeechSynthesizer` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open the source file **helloworld.cpp**.
2. Replace all the code with the following snippet:

```

// Creates an instance of a speech config with specified subscription key and service region.
// Replace with your own subscription key and service region (e.g., "westus").
auto config = SpeechConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");

// Creates a speech synthesizer using file as audio output.
// Replace with your own audio file name.
auto fileName = "helloworld.wav";
auto fileOutput = AudioConfig::FromWavFileOutput(fileName);
auto synthesizer = SpeechSynthesizer::FromConfig(config, fileOutput);

// Converts the specified text to speech, saving the audio data in the file specified above.
// Replace with your own text.
auto text = "Hello world!";
auto result = synthesizer->SpeakTextAsync(text).get();

// Checks result for successful completion or errors.
if (result->Reason == ResultReason::SynthesizingAudioCompleted)
{
    cout << "Speech synthesized to [" << fileName << "] for text [" << text << "]" << std::endl;
}
else if (result->Reason == ResultReason::Canceled)
{
    auto cancellation = SpeechSynthesisCancellationDetails::FromResult(result);
    cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

    if (cancellation->Reason == CancellationReason::Error)
    {
        cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
        cout << "CANCELED: ErrorDetails=[" << cancellation->ErrorDetails << "]" << std::endl;
        cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}

```

3. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
4. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
5. Replace the string `helloworld.wav` with your own filename.
6. From the menu bar, choose **File > Save All**.

Build and run the application

1. From the menu bar, select **Build > Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug > Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Your text is converted to speech, and saved in the audio data specified.

Speech synthesized to [helloworld.wav] for text [Hello world!]

Next steps

[Explore C++ samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech in an audio file. After satisfying a few prerequisites, synthesizing speech into a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an Audio Configuration object that specifies the .WAV file name.
- Create a `SpeechSynthesizer` object using the configuration objects from above.
- Using the `SpeechSynthesizer` object, convert your text into synthesized speech, saving it into the audio file specified.
- Inspect the `SpeechSynthesizer` returned for errors.

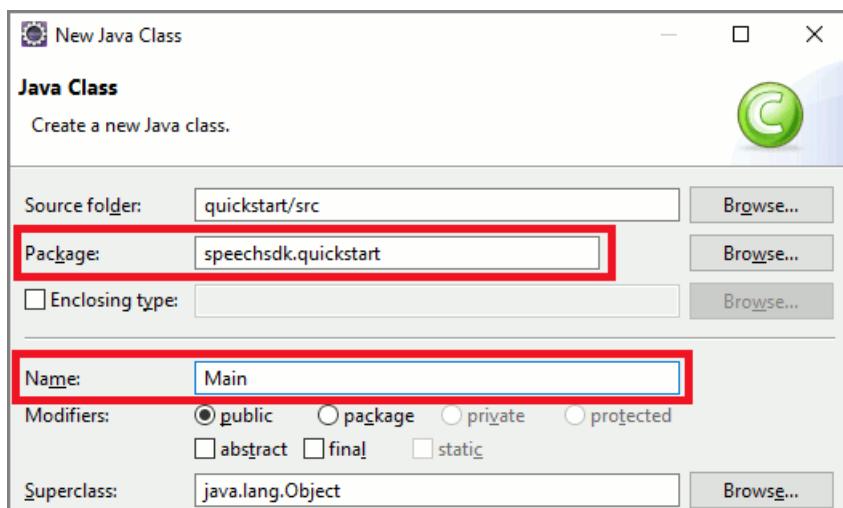
If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in `Main.java` with the following snippet:

```
package speechsdk.quickstart;

import java.util.concurrent.Future;
import com.microsoft.cognitiveservices.speech.*;

/**
 * Quickstart: recognize speech using the Speech SDK for Java.
 */
public class Main {

    /**
     *

```

```

* @param args Arguments are ignored in this sample.
*/
public static void main(String[] args) {
    try {
        // Replace below with your own subscription key
        String speechSubscriptionKey = "YourSubscriptionKey";
        // Replace below with your own service region (e.g., "westus").
        String serviceRegion = "YourServiceRegion";
        // Replace below with your own filename.
        String audioFileName = "helloworld.wav";
        // Replace below with your own filename.
        String text = "Hello world!";

        int exitCode = 1;
        SpeechConfig config = SpeechConfig.fromSubscription(speechSubscriptionKey, serviceRegion);
        assert(config != null);

        AudioConfig audioOutput = AudioConfig.fromWavFileInput(audioFileName);
        assert(audioOutput != null);

        SpeechSynthesizer synth = new SpeechSynthesizer(config, audioOutput);
        assert(synth != null);

        Future<SpeechSynthesisResult> task = synth.SpeakTextAsync(text);
        assert(task != null);

        SpeechSynthesisResult result = task.get();
        assert(result != null);

        if (result.getReason() == ResultReason.SynthesizingAudioCompleted) {
            System.out.println("Speech synthesized to [" + audioFileName + "] for text [" + text +
                "]");
            exitCode = 0;
        }
        else if (result.getReason() == ResultReason.Canceled) {
            SpeechSynthesisCancellationDetails cancellation =
                SpeechSynthesisCancellationDetails.fromResult(result);
            System.out.println("CANCELED: Reason=" + cancellation.getReason());

            if (cancellation.getReason() == CancellationReason.Error) {
                System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
                System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
                System.out.println("CANCELED: Did you update the subscription info?");
            }
        }
        result.close();
        synth.close();

        System.exit(exitCode);
    } catch (Exception ex) {
        System.out.println("Unexpected exception: " + ex.getMessage());

        assert(false);
        System.exit(1);
    }
}
}

```

4. Replace the string `YourSubscriptionKey` with your subscription key.
5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
6. Replace the string `helloworld.wav` with your own filename.
7. Replace the string `Hello world!` with your own text.

8. Save changes to the project.

Build and run the app

Press F11, or select **Run > Debug**. Your text is converted to speech, and saved in the audio data specified.

```
Speech synthesized to [helloworld.wav] for text [Hello world!]
```

Next steps

[Explore Java samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech in an audio file. After satisfying a few prerequisites, synthesizing speech into a file only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an Audio Configuration object that specifies the .WAV file name.
- Create a `SpeechSynthesizer` object using the configuration objects from above.
- Using the `SpeechSynthesizer` object, convert your text into synthesized speech, saving it into the audio file specified.
- Inspect the `SpeechSynthesizer` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

- An Azure subscription key for the Speech service. [Get one for free](#).
- [Python 3.5 or later](#).
- The Python Speech SDK package is available for these operating systems:
 - Windows: x64 and x86.
 - Mac: macOS X version 10.12 or later.
 - Linux: Ubuntu 16.04, Ubuntu 18.04, Debian 9 on x64.
- On Linux, run these commands to install the required packages:
 - On Ubuntu:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libasound2
```
 - On Debian 9:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.2 libasound2
```

- On Windows, you need the [Microsoft Visual C++ Redistributable for Visual Studio 2019](#) for your platform.

Install the Speech SDK

IMPORTANT

By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

This command installs the Python package from PyPI for the Speech SDK:

```
pip install azure-cognitiveservices-speech
```

Support and updates

Updates to the Speech SDK Python package are distributed via PyPI and announced in the [Release notes](#). If a new version is available, you can update to it with the command

```
pip install --upgrade azure-cognitiveservices-speech
```

. Check which version is currently installed by inspecting the `azure.cognitiveservices.speech._version_` variable.

If you have a problem, or you're missing a feature, see [Support and help options](#).

Create a Python application that uses the Speech SDK

Run the sample

You can copy the [sample code](#) from this quickstart to a source file `quickstart.py` and run it in your IDE or in the console:

```
python quickstart.py
```

Or you can download this quickstart tutorial as a [Jupyter](#) notebook from the [Speech SDK sample repository](#) and run it as a notebook.

Sample code

```

import azure.cognitiveservices.speech as speechsdk

# Creates an instance of a speech config with specified subscription key and service region.
# Replace with your own subscription key and service region (e.g., "westus").
speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"
speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)

# Creates an audio configuration that points to an audio file.
# Replace with your own audio filename.
audio_filename = "helloworld.wav"
audio_output = speechsdk.AudioOutputConfig(filename=audio_filename)

# Creates a synthesizer with the given settings
speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config, audio_config=audio_output)

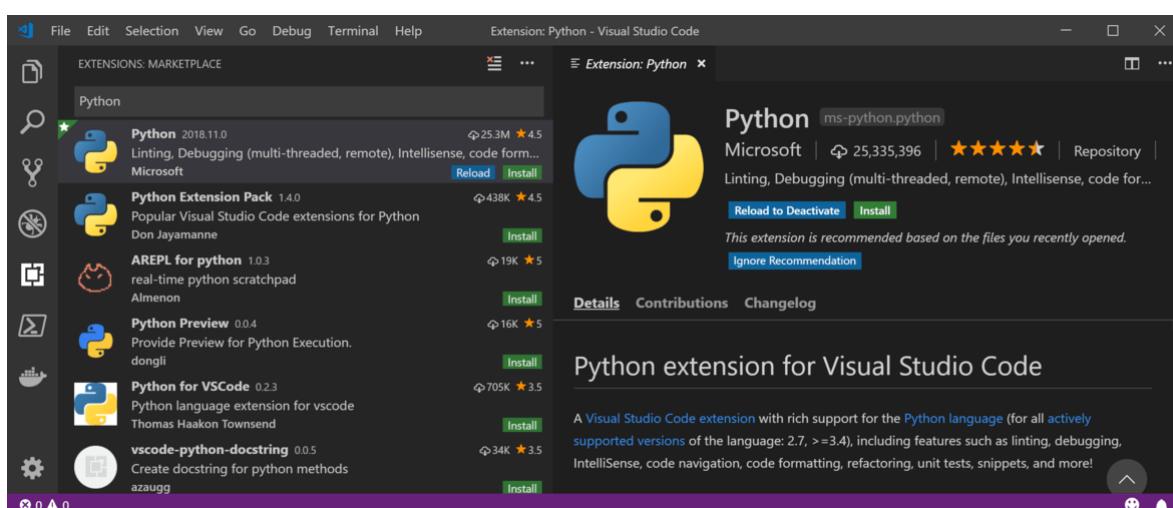
# Synthesizes the text to speech.
# Replace with your own text.
text = "Hello world!"
result = speech_synthesizer.speak_text_async(text).get()

# Checks result.
if result.reason == speechsdk.ResultReason.SynthesizingAudioCompleted:
    print("Speech synthesized to [{}]\nfor text [{}].format(audio_filename, text))")
elif result.reason == speechsdk.ResultReason.Canceled:
    cancellation_details = result.cancellation_details
    print("Speech synthesis canceled: {}.\nformat(cancellation_details.reason))")
    if cancellation_details.reason == speechsdk.CancellationReason.Error:
        if cancellation_details.error_details:
            print("Error details: {}.\nformat(cancellation_details.error_details))")
    print("Did you update the subscription info?")

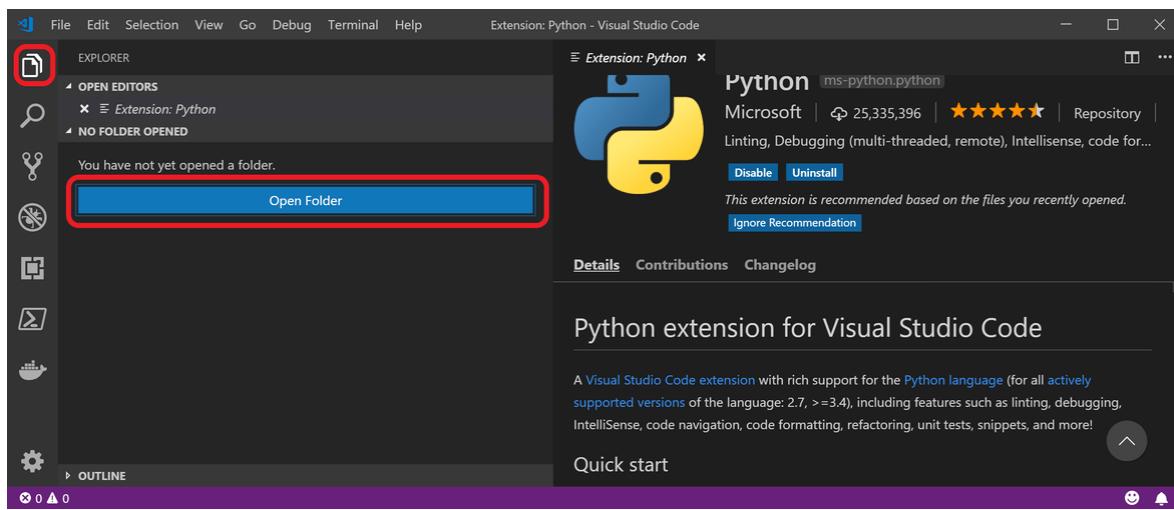
```

Install and use the Speech SDK with Visual Studio Code

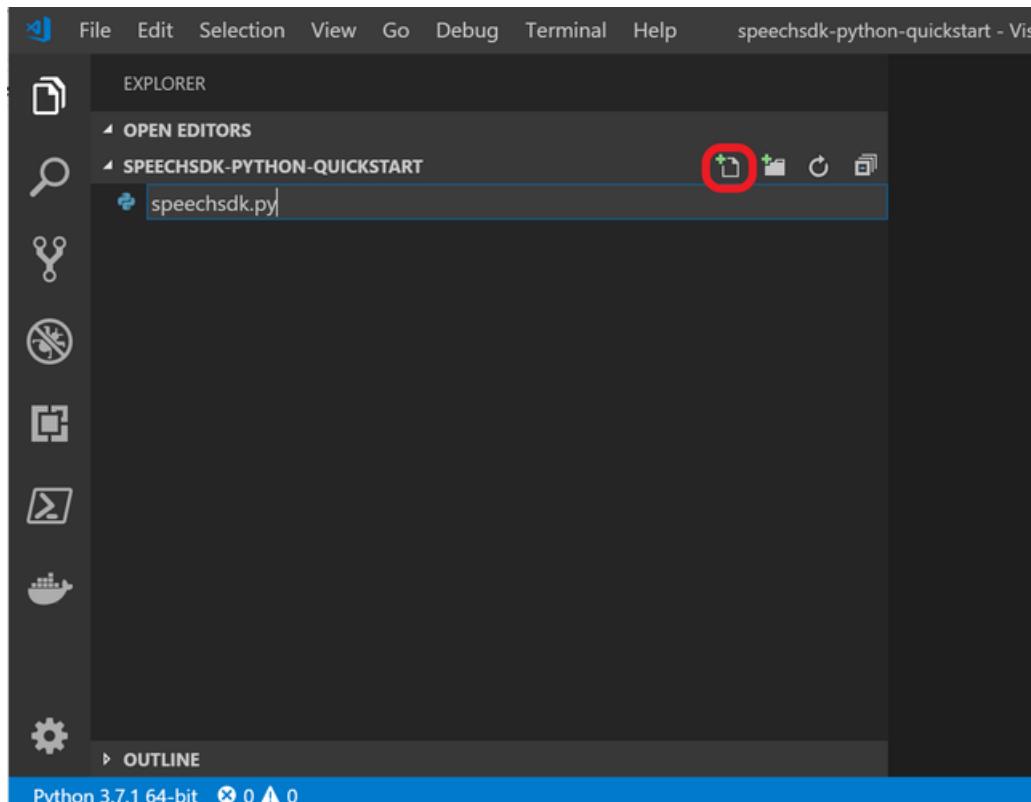
1. Download and install a 64-bit version of [Python](#), 3.5 or later, on your computer.
2. Download and install [Visual Studio Code](#).
3. Open Visual Studio Code and install the Python extension. Select **File > Preferences > Extensions** from the menu. Search for **Python**.



4. Create a folder to store the project in. An example is by using Windows Explorer.
5. In Visual Studio Code, select the **File** icon. Then open the folder you created.



6. Create a new Python source file, `speechsdk.py`, by selecting the new file icon.



7. Copy, paste, and save the [Python code](#) to the newly created file.
8. Insert your Speech service subscription information.
9. If selected, a Python interpreter displays on the left side of the status bar at the bottom of the window. Otherwise, bring up a list of available Python interpreters. Open the command palette (Ctrl+Shift+P) and enter **Python: Select Interpreter**. Choose an appropriate one.
10. You can install the Speech SDK Python package from within Visual Studio Code. Do that if it's not installed yet for the Python interpreter you selected. To install the Speech SDK package, open a terminal. Bring up the command palette again (Ctrl+Shift+P) and enter **Terminal: Create New Integrated Terminal**. In the terminal that opens, enter the command
`python -m pip install azure-cognitiveservices-speech` or the appropriate command for your system.
11. To run the sample code, right-click somewhere inside the editor. Select **Run Python File in Terminal**. Your text is converted to speech, and saved in the audio data specified.

Speech synthesized to [helloworld.wav] for text [Hello world!]

If you have issues following these instructions, refer to the more extensive [Visual Studio Code Python tutorial](#).

Next steps

[Explore Python samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	ADDITIONAL QUICKSTARTS	CODE SAMPLES
C++		Windows , Linux , macOS
C#		.NET Framework , .NET Core , UWP , Unity , Xamarin
Java		Android , JRE
Javascript		Browser
Node.js		Windows , Linux , macOS
Objective-C	macOS , iOS	iOS , macOS
Python		Windows , Linux , macOS
Swift	macOS , iOS	iOS , macOS

Quickstart: Synthesize speech to a speaker

12/4/2019 • 28 minutes to read • [Edit Online](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech. After satisfying a few prerequisites, rendering synthesized speech to the default speakers only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechSynthesizer` object using the `SpeechConfig` object from above.
- Using the `SpeechSynthesizer` object to speak the text.
- Check the `SpeechSynthesisResult` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [.NET](#)
- [.NET Core](#)
- [Unity](#)
- [UWP](#)

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open **Program.cs** and replace the automatically generated code with this sample:

```

using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;

namespace helloworld
{
    class Program
    {
        public static async Task SynthesisToSpeakerAsync()
        {
            // Creates an instance of a speech config with specified subscription key and service
            region.

            // Replace with your own subscription key and service region (e.g., "westus").
            // The default language is "en-us".
            var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");

            // Creates a speech synthesizer using the default speaker as audio output.
            using (var synthesizer = new SpeechSynthesizer(config))
            {
                // Receive a text from console input and synthesize it to speaker.
                Console.WriteLine("Type some text that you want to speak...");
                Console.Write("> ");
                string text = Console.ReadLine();

                using (var result = await synthesizer.SpeakTextAsync(text))
                {
                    if (result.Reason == ResultReason.SynthesizingAudioCompleted)
                    {
                        Console.WriteLine($"Speech synthesized to speaker for text [{text}]");
                    }
                    else if (result.Reason == ResultReason.Canceled)
                    {
                        var cancellation = SpeechSynthesisCancellationDetails.FromResult(result);
                        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

                        if (cancellation.Reason == CancellationReason.Error)
                        {
                            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                            Console.WriteLine($"CANCELED: ErrorDetails=");
                            [cancellation.ErrorDetails]);
                            Console.WriteLine($"CANCELED: Did you update the subscription info?");
                        }
                    }
                }

                // This is to give some time for the speaker to finish playing back the audio
                Console.WriteLine("Press any key to exit...");
                Console.ReadKey();
            }
        }

        static void Main()
        {
            SynthesisToSpeakerAsync().Wait();
        }
    }
}

```

2. Find the string `YourSubscriptionKey`, and replace it with your Speech service subscription key.
3. Find the string `YourServiceRegion`, and replace it with the `region` associated with your subscription. For example, if you're using the free trial subscription, the region is `westus`.
4. From the menu bar, choose **File > Save All**.

Build and run the application

1. From the menu bar, choose **Build > Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug > Start Debugging** (or select **F5**) to start the **helloworld** application.
3. Enter an English phrase or sentence. The application transmits your text to the Speech service, which sends synthesized speech to the application to play on your speaker.

```
Type some text that you want to speak...
> hello
Speech synthesized to speaker for text [hello]
Press any key to exit...
```

Next steps

[Explore C# samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech. After satisfying a few prerequisites, rendering synthesized speech to the default speakers only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechSynthesizer` object using the `SpeechConfig` object from above.
- Using the `SpeechSynthesizer` object to speak the text.
- Check the `SpeechSynthesisResult` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [Linux](#)
- [Windows](#)

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Create a C++ source file named `helloworld.cpp`, and paste the following code into it.

```

#include <iostream> // cin, cout
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;

void synthesizeSpeech()
{
    // Creates an instance of a speech config with specified subscription key and service region.
    // Replace with your own subscription key and service region (e.g., "westus").
    auto config = SpeechConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");

    // Creates a speech synthesizer using the default speaker as audio output. The default spoken
    language is "en-us".
    auto synthesizer = SpeechSynthesizer::FromConfig(config);

    // Receive a text from console input and synthesize it to speaker.
    cout << "Type some text that you want to speak..." << std::endl;
    cout << "> ";
    std::string text;
    getline(cin, text);

    auto result = synthesizer->SpeakTextAsync(text).get();

    // Checks result.
    if (result->Reason == ResultReason::SynthesizingAudioCompleted)
    {
        cout << "Speech synthesized to speaker for text [" << text << "]" << std::endl;
    }
    else if (result->Reason == ResultReason::Canceled)
    {
        auto cancellation = SpeechSynthesisCancellationDetails::FromResult(result);
        cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

        if (cancellation->Reason == CancellationReason::Error)
        {
            cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
            cout << "CANCELED: ErrorDetails=[ " << cancellation->ErrorDetails << " ]" << std::endl;
            cout << "CANCELED: Did you update the subscription info?" << std::endl;
        }
    }
}

// This is to give some time for the speaker to finish playing back the audio
cout << "Press enter to exit..." << std::endl;
cin.get();
}

int main(int argc, char **argv) {
    setlocale(LC_ALL, "");
    synthesizeSpeech();
    return 0;
}

```

2. In this new file, replace the string `YourSubscriptionKey` with your Speech service subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).

Build the app

NOTE

Make sure to enter the commands below as a *single command line*. The easiest way to do that is to copy the command by using the **Copy** button next to each command, and then paste it at your shell prompt.

- On an **x64** (64-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I "$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L "$SPEECHSDK_ROOT/lib/x64" -l:libasound.so.2
```

- On an **x86** (32-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I "$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L "$SPEECHSDK_ROOT/lib/x86" -l:libasound.so.2
```

- On an **ARM64** (64-bit) system, run the following command to build the application.

```
g++ helloworld.cpp -o helloworld -I "$SPEECHSDK_ROOT/include/cxx_api" -I "$SPEECHSDK_ROOT/include/c_api" --std=c++14 -lpthread -lMicrosoft.CognitiveServices.Speech.core -L "$SPEECHSDK_ROOT/lib/arm64" -l:libasound.so.2
```

Run the app

- Configure the loader's library path to point to the Speech SDK library.

- On an **x64** (64-bit) system, enter the following command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/x64"
```

- On an **x86** (32-bit) system, enter this command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/x86"
```

- On an **ARM64** (64-bit) system, enter the following command.

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$SPEECHSDK_ROOT/lib/arm64"
```

- Run the application.

```
./helloworld
```

- In the console window, a prompt appears, prompting you to type some text. Type a few words or a sentence. The text that you typed is transmitted to the Speech service and synthesized to speech, which plays on your speaker.

```
Type some text that you want to speak...
> hello
Speech synthesized to speaker for text [hello]
Press enter to exit...
```

Next steps

[Explore C++ samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech. After satisfying a few prerequisites, rendering synthesized speech to the default speakers only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create a `SpeechSynthesizer` object using the `SpeechConfig` object from above.
- Using the `SpeechSynthesizer` object to speak the text.
- Check the `SpeechSynthesisResult` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [Java Runtime](#)
- [Android](#)

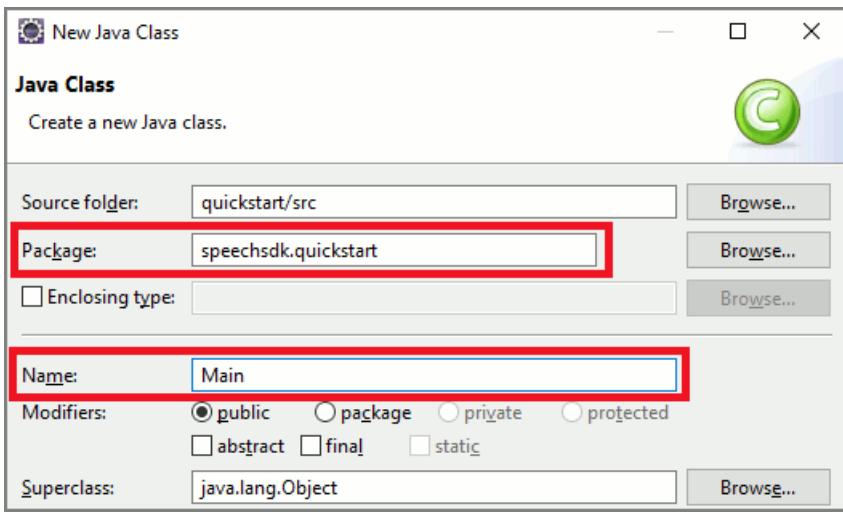
Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in `Main.java` with the following snippet:

```
package speechsdk.quickstart;

import java.util.Scanner;
import java.util.concurrent.Future;
import com.microsoft.cognitiveservices.speech.*;

/**
 * Quickstart: synthesize speech using the Speech SDK for Java.
 */
public class Main {

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {
        try {
            // Replace below with your own subscription key
            String speechSubscriptionKey = "YourSubscriptionKey";
            // Replace below with your own service region (e.g., "westus").
            String serviceRegion = "YourServiceRegion";

            int exitCode = 1;
            SpeechConfig config = SpeechConfig.fromSubscription(speechSubscriptionKey,
serviceRegion);
            assert(config != null);

            SpeechSynthesizer synth = new SpeechSynthesizer(config);
            assert(synth != null);

            System.out.println("Type some text that you want to speak...");
            System.out.print("> ");
            String text = new Scanner(System.in).nextLine();

            Future<SpeechSynthesisResult> task = synth.SpeakTextAsync(text);
            assert(task != null);

            SpeechSynthesisResult result = task.get();
            assert(result != null);

            if (result.getReason() == ResultReason.SynthesizingAudioCompleted) {
                System.out.println("Speech synthesized to speaker for text [" + text + "]");
                exitCode = 0;
            }
            else if (result.getReason() == ResultReason.Canceled) {
                SpeechSynthesisCancellationDetails cancellation =
SpeechSynthesisCancellationDetails.fromResult(result);
                System.out.println("CANCELED: Reason=" + cancellation.getReason());
            }
        }
    }
}
```

```
        if (cancellation.getReason() == CancellationReason.Error) {
            System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
            System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
            System.out.println("CANCELED: Did you update the subscription info?");
        }
    }

    result.close();
    synth.close();

    System.exit(exitCode);
} catch (Exception ex) {
    System.out.println("Unexpected exception: " + ex.getMessage());

    assert(false);
    System.exit(1);
}
}
```

4. Replace the string `YourSubscriptionKey` with your subscription key.
 5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
 6. Save changes to the project.

Build and run the app

Press F11, or select **Run > Debug**. Input a text when prompted, and you will hear the synthesized audio played from default speaker.

Next steps

Explore Java samples on GitHub

See also

- Create a Custom Voice
 - Record custom voice samples

In this quickstart you will use the [Speech SDK](#) to convert text to synthesized speech. After satisfying a few prerequisites, rendering synthesized speech to the default speakers only takes four steps:

- Create a `SpeechConfig` object from your subscription key and region.
 - Create a `SpeechSynthesizer` object using the `SpeechConfig` object from above.
 - Using the `SpeechSynthesizer` object to speak the text.
 - Check the `SpeechSynthesisResult` returned for errors.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- Create an Azure Speech Resource
 - Setup your development environment

- Create an empty sample project

```
## Support and updates

Updates to the Speech SDK Python package are distributed via PyPI and announced in the [Release notes](~/articles/cognitive-services/Speech-Service/releasenotes.md).
If a new version is available, you can update to it with the command `pip install --upgrade azure-cognitiveservices-speech`.
Check which version is currently installed by inspecting the `azure.cognitiveservices.speech._version_` variable.

If you have a problem, or you're missing a feature, see [Support and help options](~/articles/cognitive-services/Speech-Service/support.md).

## Create a Python application that uses the Speech SDK

### Run the sample

You can copy the [sample code](#sample-code) from this quickstart to a source file `quickstart.py` and run it in your IDE or in the console:

```sh
python quickstart.py
```

```

Or you can download this quickstart tutorial as a [Jupyter](#) notebook from the [Speech SDK sample repository](#) and run it as a notebook.

Sample code

```
import azure.cognitiveservices.speech as speechsdk

# Creates an instance of a speech config with specified subscription key and service region.
# Replace with your own subscription key and service region (e.g., "westus").
speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"
speech_config = speechsdk.SpeechConfig(subscription=speech_key, region=service_region)

# Creates a speech synthesizer using the default speaker as audio output.
speech_synthesizer = speechsdk.SpeechSynthesizer(speech_config=speech_config)

# Receives a text from console input.
print("Type some text that you want to speak...")
text = input()

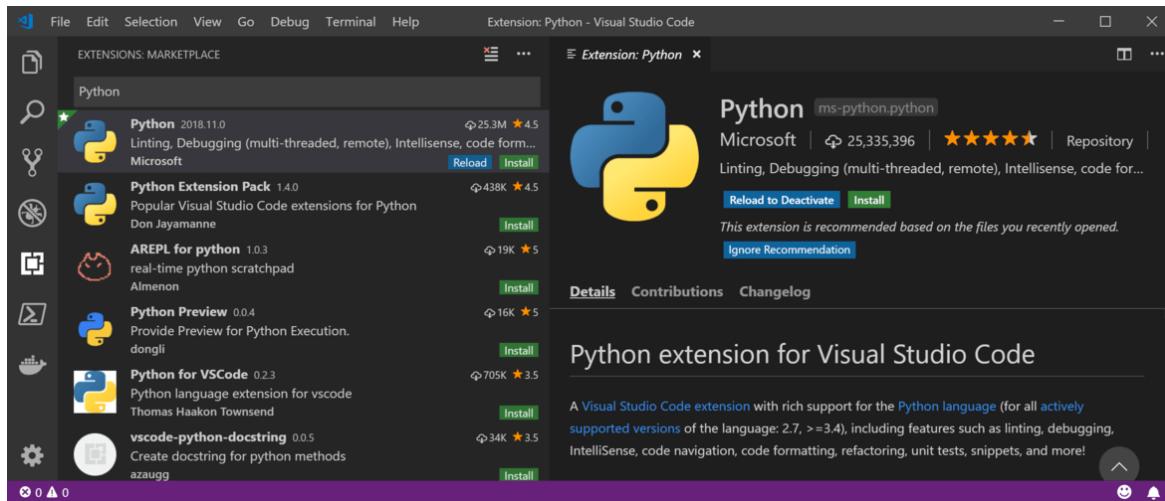
# Synthesizes the received text to speech.
# The synthesized speech is expected to be heard on the speaker with this line executed.
result = speech_synthesizer.speak_text_async(text).get()

# Checks result.
if result.reason == speechsdk.ResultReason.SynthesizingAudioCompleted:
    print("Speech synthesized to speaker for text [{}].format(text))")
elif result.reason == speechsdk.ResultReason.Canceled:
    cancellation_details = result.cancellation_details
    print("Speech synthesis canceled: {}".format(cancellation_details.reason))
    if cancellation_details.reason == speechsdk.CancellationReason.Error:
        if cancellation_details.error_details:
            print("Error details: {}".format(cancellation_details.error_details))
    print("Did you update the subscription info?")
```

Install and use the Speech SDK with Visual Studio Code

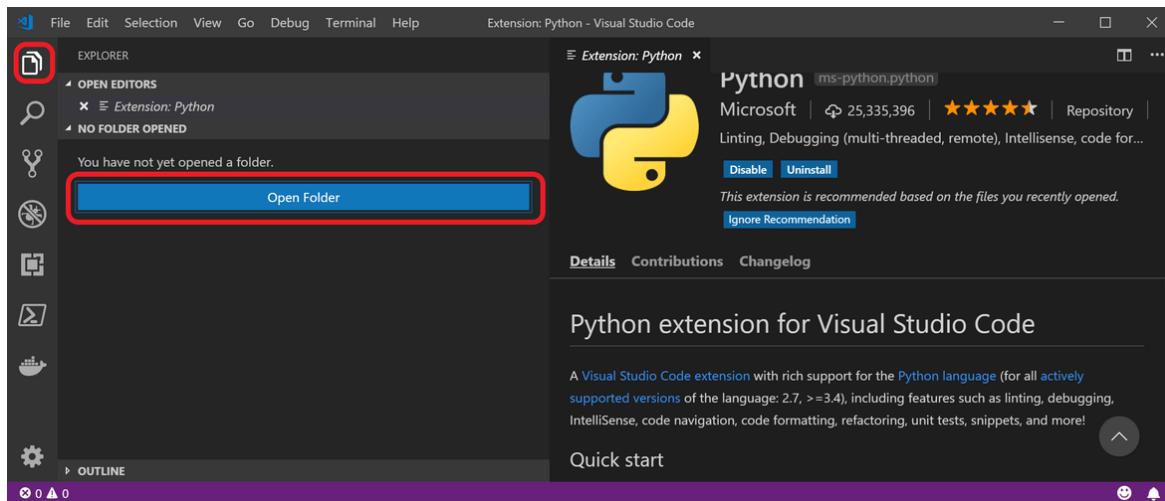
1. Download and install a 64-bit version of [Python](#), 3.5 or later, on your computer.
2. Download and install [Visual Studio Code](#).

3. Open Visual Studio Code and install the Python extension. Select **File > Preferences > Extensions** from the menu. Search for **Python**.

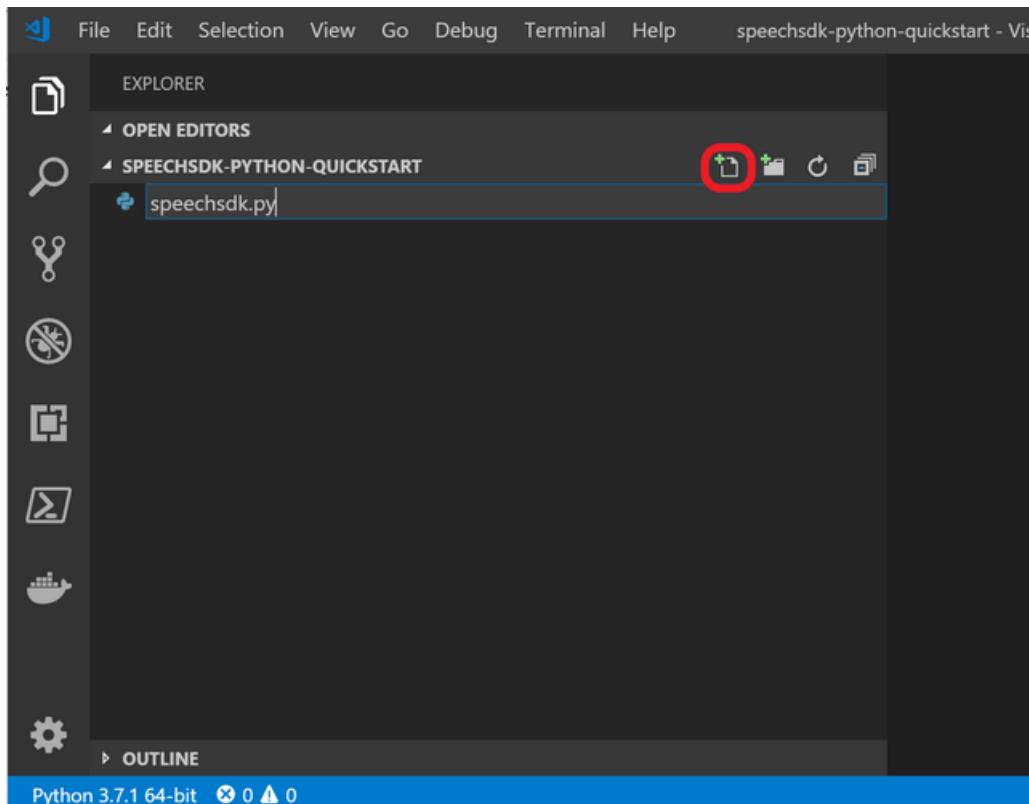


4. Create a folder to store the project in. An example is by using Windows Explorer.

5. In Visual Studio Code, select the **File** icon. Then open the folder you created.



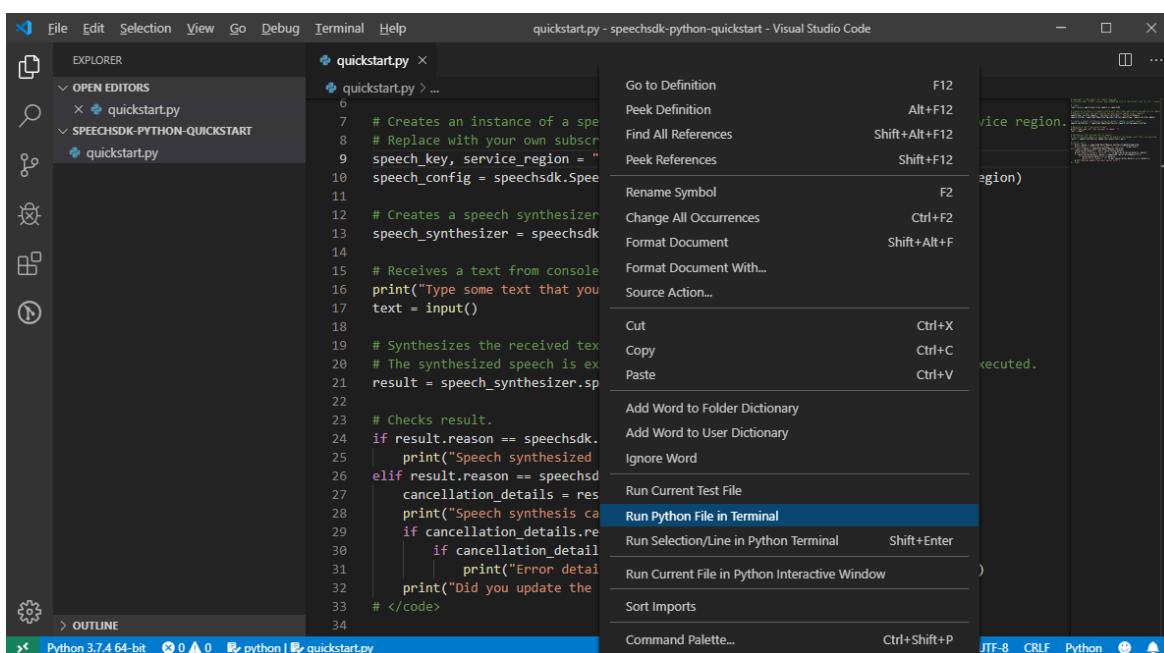
6. Create a new Python source file, `speechsdk.py`, by selecting the new file icon.



7. Copy, paste, and save the [Python code](#) to the newly created file.
8. Insert your Speech service subscription information.
9. If selected, a Python interpreter displays on the left side of the status bar at the bottom of the window. Otherwise, bring up a list of available Python interpreters. Open the command palette (Ctrl+Shift+P) and enter **Python: Select Interpreter**. Choose an appropriate one.
10. You can install the Speech SDK Python package from within Visual Studio Code. Do that if it's not installed yet for the Python interpreter you selected. To install the Speech SDK package, open a terminal. Bring up the command palette again (Ctrl+Shift+P) and enter **Terminal: Create New Integrated Terminal**. In the terminal that opens, enter the command


```
python -m pip install azure-cognitiveservices-speech
```

 or the appropriate command for your system.
11. To run the sample code, right-click somewhere inside the editor. Select **Run Python File in Terminal**. Type some text when you're prompted. The synthesized audio is played shortly afterward.



If you have issues following these instructions, refer to the more extensive [Visual Studio Code Python tutorial](#).

Next steps

[Explore Python samples on GitHub](#)

See also

- [Create a Custom Voice](#)
- [Record custom voice samples](#)

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

| LANGUAGE | ADDITIONAL QUICKSTARTS | CODE SAMPLES |
|-------------|---|--|
| C++ | | Quickstarts , Samples |
| C# | | .NET Framework , .NET Core , UWP , Unity , Xamarin |
| Java | | Android , JRE |
| Javascript | | Browser |
| Node.js | | Windows , Linux , macOS |
| Objective-C | macOS , iOS | iOS , macOS |
| Python | | Windows , Linux , macOS |
| Swift | macOS , iOS | iOS , macOS |

Quickstart: Asynchronous synthesis for long-form audio in Python (Preview)

12/4/2019 • 6 minutes to read • [Edit Online](#)

In this quickstart, you'll use the Long Audio API to asynchronously convert text to speech, and retrieve the audio output from a URI provided by the service. This REST API is ideal for content providers that need to convert text files greater than 10,000 characters or 50 paragraphs into synthesized speech. For more information, see [Long Audio API](#).

NOTE

Asynchronous synthesis for long-form audio can only be used with [Custom Neural Voices](#).

Prerequisites

This quickstart requires:

- Python 2.7.x or 3.x.
- [Visual Studio](#), [Visual Studio Code](#), or your favorite text editor.
- An Azure subscription and a Speech service subscription key. [Create an Azure account](#) and [create a speech resource](#) to get the key. When creating the Speech resource, make sure that your pricing tier is set to **S0**, and location is set to a [supported region](#).

Create a project and import required modules

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

`voice_synthesis_client.py`.

```
import argparse
import json
import ntpath
import urllib3
import requests
import time
from json import dumps, loads, JSONEncoder, JSONDecoder
import pickle

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

NOTE

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

`pip install requests urllib3`.

These modules are used to parse arguments, construct the HTTP request, and call the text-to-speech long audio REST API.

Get a list of supported voices

This code gets a list of available voices that you can use to convert text-to-speech. Add this code

`voice_synthesis_client.py :`

```
parser = argparse.ArgumentParser(description='Cris client tool to submit voice synthesis requests.')
parser.add_argument('--voices', action="store_true", default=False, help='print voice list')
parser.add_argument('-key', action="store", dest="key", required=True, help='the cris subscription key, like ff1eb62d06d34767bda0207acb1da7d7 ')
parser.add_argument('-region', action="store", dest="region", required=True, help='the region information, could be centralindia, canadacentral or uksouth')
args = parser.parse_args()
baseAddress = 'https://%s.cris.ai/api/texttospeech/v3.0-beta1/' % args.region

def getVoices():
    response=requests.get(baseAddress+"voicesynthesis/voices", headers={"Ocp-Apim-Subscription-Key":args.key}, verify=False)
    voices = json.loads(response.text)
    return voices

if args.voices:
    voices = getVoices()
    print("There are %d voices available:" % len(voices))
    for voice in voices:
        print ("Name: %s, Description: %s, Id: %s, Locale: %s, Gender: %s, PublicVoice: %s, Created: %s" %
(voice['name'], voice['description'], voice['id'], voice['locale'], voice['gender'], voice['isPublicVoice'], voice['created']))
```

Test your code

Let's test what you've done so far. Run this command, replacing `<your_key>` with your Speech subscription key, and `<region>` with the region where your Speech resource was created (for example: `eastus` or `westus`). This information is available in the **Overview** tab for your resource in the [Azure portal](#).

```
python voice_synthesis_client.py --voices -key <your_key> -region <Region>
```

You should get an output that looks like this:

```
There are xx voices available:
```

```
Name: Microsoft Server Speech Text to Speech Voice (en-US, xxx), Description: xxx , Id: xxx, Locale: en-US, Gender: Male, PublicVoice: xxx, Created: 2019-07-22T09:38:14Z
Name: Microsoft Server Speech Text to Speech Voice (zh-CN, xxx), Description: xxx , Id: xxx, Locale: zh-CN, Gender: Female, PublicVoice: xxx, Created: 2019-08-26T04:55:39Z
```

Convert text to speech

The next step is to prepare an input text file. It can be either plain text or SSML, but must be more than 10,000 character or 50 paragraphs. For a complete list of requirements, see [Long Audio API](#).

After you've prepared the text file. The next step is to add code for speech synthesis to your project. Add this code to `voice_synthesis_client.py`:

NOTE

By default, the audio output is set to riff-16khz-16bit-mono-pcm. For more information about supported audio outputs, see [Long Audio API](#).

```

parser.add_argument('--submit', action="store_true", default=False, help='submit a synthesis request')
parser.add_argument('--concatenateResult', action="store_true", default=False, help='If concatenate result in a single wave file')
parser.add_argument('-file', action="store", dest="file", help='the input text script file path')
parser.add_argument('-voiceId', action="store", nargs='+', dest="voiceId", help='the id of the voice which used to synthesis')
parser.add_argument('-locale', action="store", dest="locale", help='the locale information like zh-CN/en-US')
parser.add_argument('-format', action="store", dest="format", default='riff-16khz-16bit-mono-pcm', help='the output audio format')

def submitSynthesis():
    modelList = args.voiceId
    data={'name': 'simple test', 'description': 'desc...', 'models': json.dumps(modelList), 'locale': args.locale, 'outputformat': args.format}
    if args.concatenateResult:
        properties={'ConcatenateResult': 'true'}
        data['properties'] = json.dumps(properties)
    if args.file is not None:
        scriptfilename=ntpath.basename(args.file)
        files = {'script': (scriptfilename, open(args.file, 'rb'), 'text/plain')}
        response = requests.post(baseAddress+"voicesynthesis", data, headers={"Ocp-Apim-Subscription-Key":args.key}, files=files, verify=False)
        if response.status_code == 202:
            location = response.headers['Operation-Location']
            id = location.split("/")[-1]
            print("Submit synthesis request successful")
            return id
        else:
            print("Submit synthesis request failed")
            print("response.status_code: %d" % response.status_code)
            print("response.text: %s" % response.text)
            return 0

    def getSubmittedSynthesis(id):
        response=requests.get(baseAddress+"voicesynthesis/"+id, headers={"Ocp-Apim-Subscription-Key":args.key}, verify=False)
        synthesis = json.loads(response.text)
        return synthesis

    if args.submit:
        id = submitSynthesis()
        if (id == 0):
            exit(1)

        while(1):
            print("\r\nChecking status")
            synthesis=getSubmittedSynthesis(id)
            if synthesis['status'] == "Succeeded":
                r = requests.get(synthesis['resultsUrl'])
                filename=id + ".zip"
                with open(filename, 'wb') as f:
                    f.write(r.content)
                print("Succeeded... Result file downloaded : " + filename)
                break
            elif synthesis['status'] == "Failed":
                print("Failed...")
                break
            elif synthesis['status'] == "Running":
                print("Running...")
            elif synthesis['status'] == "NotStarted":
                print("NotStarted...")
            time.sleep(10)

```

Test your code

Let's try making a request to synthesize text using your input file as a source. You'll need to update a few things in

the request below:

- Replace <your_key> with your Speech service subscription key. This information is available in the **Overview** tab for your resource in the [Azure portal](#).
- Replace <region> with the region where your Speech resource was created (for example: `eastus` or `westus`). This information is available in the **Overview** tab for your resource in the [Azure portal](#).
- Replace <input> with the path to the text file you're looking to convert from text-to-speech.
- Replace <locale> with the desired output locale. For more information, see [language support](#).
- Replace <voice_guid> with the desired voice for the audio output. Use one of the voices returned by [Get a list of supported voices](#) or use the list of neural voices provided in [language support](#).

Convert text to speech with this command:

```
python voice_synthesis_client.py --submit -key <your_key> -region <Region> -file <input> -locale <locale> -voiceId <voice_guid>
```

NOTE

'concatenateResult' is an optional parameter, if this parameter isn't provided, the output will be provided as multiple wave files, one for each line.

You should get an output that looks like this:

```
Submit synthesis request successful

Checking status
NotStarted...

Checking status
Running...

Checking status
Running...

Checking status
Succeeded... Result file downloaded : xxxx.zip
```

The result provided contains the input text and the audio output files generated by the service. These are downloaded as a zip.

Remove previous requests

There is a limit of 2,000 requests for each subscription. As such, there will be times that you need to remove previously submitted requests before you can make new ones. If you don't remove existing requests, you'll receive an error when you exceed 2,000.

Add this code to `voice_synthesis_client.py`:

```

parser.add_argument('--syntheses', action="store_true", default=False, help='print synthesis list')
parser.add_argument('--delete', action="store_true", default=False, help='delete a synthesis request')
parser.add_argument('-synthesisId', action="store", nargs='+', dest="synthesisId", help='the id of the voice
synthesis which need to be deleted')

def getSubmittedSynthesizes():
    response=requests.get(baseAddress+"voicesynthesis", headers={"Ocp-Apim-Subscription-Key":args.key},
verify=False)
    syntheses = json.loads(response.text)
    return syntheses

def deleteSynthesis(ids):
    for id in ids:
        print("delete voice synthesis %s " % id)
        response = requests.delete(baseAddress+"voicesynthesis/"+id, headers={"Ocp-Apim-Subscription-
Key":args.key}, verify=False)
        if (response.status_code == 204):
            print("delete successful")
        else:
            print("delete failed, response.status_code: %d, response.text: %s " % (response.status_code,
response.text))

if args.syntheses:
    synthese = getSubmittedSynthesizes()
    print("There are %d synthesis requests submitted:" % len(synthese))
    for synthesis in synthese:
        print ("ID : %s , Name : %s, Status : %s " % (synthesis['id'], synthesis['name'],
synthesis['status']))

if args.delete:
    deleteSynthesis(args.synthesisId)

```

Test your code

Run this command, replacing <your_key> with your Speech subscription key, and <region> with the region where your Speech resource was created (for example: `eastus` or `westus`). This information is available in the **Overview** tab for your resource in the [Azure portal](#).

```
python voice_synthesis_client.py - syntheses -key <your_key> -region <Region>
```

This will return a list of syntheses you've requested. You should get an output that looks like this:

```
There are <number> synthesis requests submitted:
ID : xxx , Name : xxx, Status : Succeeded
ID : xxx , Name : xxx, Status : Running
ID : xxx , Name : xxx : Succeeded
```

Now let's use some of these values to remove/delete previously submitted requests. Run this command, replacing <your_key> with your Speech subscription key, and <region> with the region where your Speech resource was created (for example: `eastus` or `westus`). This information is available in the **Overview** tab for your resource in the [Azure portal](#). The <synthesis_id> should be one of the values returned in the previous request.

NOTE

Requests with a status of 'Running'/'Waiting' cannot be removed or deleted.

```
python voice_synthesis_client.py - delete -key <your_key> -region <Region> -synthesisId <synthesis_id>
```

You should get an output that looks like this:

```
delete voice synthesis xxx
delete successful
```

Get the full client

The complete `voice_synthesis_client.py` is available for download on [GitHub](#).

Next steps

[Learn more about the Long Audio API](#)

Speech Synthesis Markup Language (SSML)

12/31/2019 • 15 minutes to read • [Edit Online](#)

Speech Synthesis Markup Language (SSML) is an XML-based markup language that lets developers specify how input text is converted into synthesized speech using the text-to-speech service. Compared to plain text, SSML allows developers to fine-tune the pitch, pronunciation, speaking rate, volume, and more of the text-to-speech output. Normal punctuation, such as pausing after a period, or using the correct intonation when a sentence ends with a question mark are automatically handled.

The Speech service implementation of SSML is based on World Wide Web Consortium's [Speech Synthesis Markup Language Version 1.0](#).

IMPORTANT

Chinese, Japanese, and Korean characters count as two characters for billing. For more information, see [Pricing](#).

Standard, neural, and custom voices

Choose from standard and neural voices, or create your own custom voice unique to your product or brand. 75+ standard voices are available in more than 45 languages and locales, and 5 neural voices are available in 4 languages and locales. For a complete list of supported languages, locales, and voices (neural and standard), see [language support](#).

To learn more about standard, neural, and custom voices, see [Text-to-speech overview](#).

Special characters

While using SSML to convert text-to-synthesized speech, keep in mind that just like with XML, special characters, such as quotation marks, apostrophes, and brackets must be escaped. For more information, see [Extensible Markup Language \(XML\) 1.0: Appendix D](#).

Supported SSML elements

Each SSML document is created with SSML elements (or tags). These elements are used to adjust pitch, prosody, volume, and more. The following sections detail how each element is used, and when an element is required or optional.

IMPORTANT

Don't forget to use double quotes around attribute values. Standards for well-formed, valid XML requires attribute values to be enclosed in double quotation marks. For example, `<prosody volume="90">` is a well-formed, valid element, but `<prosody volume=90>` is not. SSML may not recognize attribute values that are not in quotes.

Create an SSML document

`speak` is the root element, and is **required** for all SSML documents. The `speak` element contains important information, such as version, language, and the markup vocabulary definition.

Syntax

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="string"></speak>
```

Attributes

| ATTRIBUTE | DESCRIPTION | REQUIRED / OPTIONAL |
|-----------|---|---------------------|
| version | Indicates the version of the SSML specification used to interpret the document markup. The current version is 1.0. | Required |
| xml:lang | Specifies the language of the root document. The value may contain a lowercase, two-letter language code (for example, en), or the language code and uppercase country/region (for example, en-US). | Required |
| xmlns | Specifies the URI to the document that defines the markup vocabulary (the element types and attribute names) of the SSML document. The current URI is https://www.w3.org/2001/10/synthesis . | Required |

Choose a voice for text-to-speech

The `voice` element is required. It is used to specify the voice that is used for text-to-speech.

Syntax

```
<voice name="string">
    This text will get converted into synthesized speech.
</voice>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
name	Identifies the voice used for text-to-speech output. For a complete list of supported voices, see Language support .	Required

Example

NOTE

This example uses the `en-US-Jessa24kRUS` voice. For a complete list of supported voices, see [Language support](#).

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    This is the text that is spoken.
  </voice>
</speak>
```

Use multiple voices

Within the `speak` element, you can specify multiple voices for text-to-speech output. These voices can be in different languages. For each voice, the text must be wrapped in a `voice` element.

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
name	Identifies the voice used for text-to-speech output. For a complete list of supported voices, see Language support .	Required

Example

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    Good morning!
  </voice>
  <voice name="en-US-Guy24kRUS">
    Good morning to you too Jessa!
  </voice>
</speak>
```

Adjust speaking styles

IMPORTANT

This feature will only work with neural voices.

By default, the text-to-speech service synthesizes text using a neutral speaking style for both standard and neural voices. With neural voices, you can adjust the speaking style to express cheerfulness, empathy, or sentiment with the `<mstts:express-as>` element. This is an optional element unique to the Speech service.

Currently, speaking style adjustments are supported for these neural voices:

- `en-US-JessaNeural`
- `zh-CN-XiaoxiaoNeural`

Changes are applied at the sentence level, and style vary by voice. If a style isn't supported, the service will return speech in the default neutral speaking style.

Syntax

```
<mstts:express-as type="string"></mstts:express-as>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
type	Specifies the speaking style. Currently, speaking styles are voice specific.	Required if adjusting the speaking style for a neural voice. If using <code>mstts:express-as</code> , then type must be provided. If an invalid value is provided, this element will be ignored.

Use this table to determine which speaking styles are supported for each neural voice.

VOICE	TYPE	DESCRIPTION
en-US-JessaNeural	type= <code>cheerful</code>	Expresses an emotion that is positive and happy
	type= <code>empathy</code>	Expresses a sense of caring and understanding
	type= <code>chat</code>	Speak in a casual, relaxed tone
	type= <code>newscast</code>	Expresses a formal tone, similar to news broadcasts
	type= <code>customerservice</code>	Speak in a friendly and patient way as customer service
zh-CN-XiaoxiaoNeural	type= <code>newscast</code>	Expresses a formal tone, similar to news broadcasts
	type= <code>sentiment</code>	Conveys a touching message or a story

Example

This SSML snippet illustrates how the `<mstts:express-as>` element is used to change the speaking style to `cheerful`.

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis"
      xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-JessaNeural">
    <mstts:express-as type="cheerful">
      That'd be just amazing!
    </mstts:express-as>
  </voice>
</speak>
```

Add or remove a break/pause

Use the `break` element to insert pauses (or breaks) between words, or prevent pauses automatically added by the text-to-speech service.

NOTE

Use this element to override the default behavior of text-to-speech (TTS) for a word or phrase if the synthesized speech for that word or phrase sounds unnatural. Set `strength` to `none` to prevent a prosodic break, which is automatically inserted by the text-to-speech service.

Syntax

```
<break strength="string" />  
<break time="string" />
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
strength	Specifies the relative duration of a pause using one of the following values: <ul style="list-style-type: none">• none• x-weak• weak• medium (default)• strong• x-strong	Optional
time	Specifies the absolute duration of a pause in seconds or milliseconds. Examples of valid values are 2s and 500	Optional

STRENGTH	DESCRIPTION
None, or if no value provided	0 ms
x-weak	250 ms
weak	500 ms
medium	750 ms
strong	1000 ms
x-strong	1250 ms

Example

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">  
    <voice name="en-US-Jessa24kRUS">  
        Welcome to Microsoft Cognitive Services <break time="100ms" /> Text-to-Speech API.  
    </voice>  
</speak>
```

Specify paragraphs and sentences

The `p` and `s` elements are used to denote paragraphs and sentences, respectively. In the absence of these elements, the text-to-speech service automatically determines the structure of the SSML document.

The `p` element may contain text and the following elements: `audio`, `break`, `phoneme`, `prosody`, `say-as`, `sub`, `mstts:express-as`, and `s`.

The `s` element may contain text and the following elements: `audio`, `break`, `phoneme`, `prosody`, `say-as`, `mstts:express-as`, and `sub`.

Syntax

```
<p></p>
<s></s>
```

Example

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    <p>
      <s>Introducing the sentence element.</s>
      <s>Used to mark individual sentences.</s>
    </p>
    <p>
      Another simple paragraph.
      Sentence structure in this paragraph is not explicitly marked.
    </p>
  </voice>
</speak>
```

Use phonemes to improve pronunciation

The `ph` element is used to for phonetic pronunciation in SSML documents. The `ph` element can only contain text, no other elements. Always provide human-readable speech as a fallback.

Phonetic alphabets are composed of phones, which are made up of letters, numbers, or characters, sometimes in combination. Each phone describes a unique sound of speech. This is in contrast to the Latin alphabet, where any letter may represent multiple spoken sounds. Consider the different pronunciations of the letter "c" in the words "candy" and "cease", or the different pronunciations of the letter combination "th" in the words "thing" and "those".

Syntax

```
<phoneme alphabet="string" ph="string"></phoneme>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
-----------	-------------	---------------------

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
alphabet	<p>Specifies the phonetic alphabet to use when synthesizing the pronunciation of the string in the <code>ph</code> attribute. The string specifying the alphabet must be specified in lowercase letters. The following are the possible alphabets that you may specify.</p> <ul style="list-style-type: none"> • ipa – International Phonetic Alphabet • sapi – Speech API Phone Set • ups – Universal Phone Set <p>The alphabet applies only to the phoneme in the element. For more information, see Phonetic Alphabet Reference.</p>	Optional
ph	A string containing phones that specify the pronunciation of the word in the <code>phoneme</code> element. If the specified string contains unrecognized phones, the text-to-speech (TTS) service rejects the entire SSML document and produces none of the speech output specified in the document.	Required if using phonemes.

Examples

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    <s>His name is Mike <phoneme alphabet="ups" ph="JH AU"> Zhou </phoneme></s>
  </voice>
</speak>
```

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    <phoneme alphabet="ipa" ph="t&#x259;mei&#x325;;&##x27E;ou&#x325;"> tomato </phoneme>
  </voice>
</speak>
```

Adjust prosody

The `prosody` element is used to specify changes to pitch, contour, range, rate, duration, and volume for the text-to-speech output. The `prosody` element may contain text and the following elements: `audio`, `break`, `p`, `phoneme`, `prosody`, `say-as`, `sub`, and `s`.

Because prosodic attribute values can vary over a wide range, the speech recognizer interprets the assigned values as a suggestion of what the actual prosodic values of the selected voice should be. The text-to-speech service limits or substitutes values that are not supported. Examples of unsupported values are a pitch of 1 MHz or a volume of 120.

Syntax

```
<prosody pitch="value" contour="value" range="value" rate="value" duration="value" volume="value"></prosody>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
pitch	<p>Indicates the baseline pitch for the text. You may express the pitch as:</p> <ul style="list-style-type: none"> • An absolute value, expressed as a number followed by "Hz" (Hertz). For example, 600Hz. • A relative value, expressed as a number preceded by "+" or "-" and followed by "Hz" or "st", that specifies an amount to change the pitch. For example: +80Hz or -2st. The "st" indicates the change unit is semitone, which is half of a tone (a half step) on the standard diatonic scale. • A constant value: <ul style="list-style-type: none"> ◦ x-low ◦ low ◦ medium ◦ high ◦ x-high ◦ default 	Optional
contour	<p>Contour isn't supported for neural voices. Contour represents changes in pitch for speech content as an array of targets at specified time positions in the speech output. Each target is defined by sets of parameter pairs. For example:</p> <pre data-bbox="612 1320 993 1372"><prosody contour="(0%,+20Hz) (10%,-2st) (40%,+10Hz)"></pre> <p>The first value in each set of parameters specifies the location of the pitch change as a percentage of the duration of the text. The second value specifies the amount to raise or lower the pitch, using a relative value or an enumeration value for pitch (see pitch).</p>	Optional
range	<p>A value that represents the range of pitch for the text. You may express range using the same absolute values, relative values, or enumeration values used to describe pitch .</p>	Optional

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
rate	<p>Indicates the speaking rate of the text. You may express <code>rate</code> as:</p> <ul style="list-style-type: none"> • A relative value, expressed as a number that acts as a multiplier of the default. For example, a value of <code>1</code> results in no change in the rate. A value of <code>.5</code> results in a halving of the rate. A value of <code>3</code> results in a tripling of the rate. • A constant value: <ul style="list-style-type: none"> ◦ <code>x-slow</code> ◦ <code>slow</code> ◦ <code>medium</code> ◦ <code>fast</code> ◦ <code>x-fast</code> ◦ <code>default</code> 	Optional
duration	The period of time that should elapse while the speech synthesis (TTS) service reads the text, in seconds or milliseconds. For example, <code>2s</code> or <code>1800ms</code> .	Optional
volume	<p>Indicates the volume level of the speaking voice. You may express the volume as:</p> <ul style="list-style-type: none"> • An absolute value, expressed as a number in the range of <code>0.0</code> to <code>100.0</code>, from <i>quietest</i> to <i>loudest</i>. For example, <code>75</code>. The default is <code>100.0</code>. • A relative value, expressed as a number preceded by <code>+</code> or <code>-</code> that specifies an amount to change the volume. For example <code>+10</code> or <code>-5.5</code>. • A constant value: <ul style="list-style-type: none"> ◦ <code>silent</code> ◦ <code>x-soft</code> ◦ <code>soft</code> ◦ <code>medium</code> ◦ <code>loud</code> ◦ <code>x-loud</code> ◦ <code>default</code> 	Optional

Change speaking rate

Speaking rate can be applied to standard voices at the word or sentence-level. Whereas speaking rate can only be applied to neural voices at the sentence level.

Example

```

<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Guy24kRUS">
    <prosody rate="+30.00%">
      Welcome to Microsoft Cognitive Services Text-to-Speech API.
    </prosody>
  </voice>
</speak>

```

Change volume

Volume changes can be applied to standard voices at the word or sentence-level. Whereas volume changes can only be applied to neural voices at the sentence level.

Example

```

<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    <prosody volume="+20.00%">
      Welcome to Microsoft Cognitive Services Text-to-Speech API.
    </prosody>
  </voice>
</speak>

```

Change pitch

Pitch changes can be applied to standard voices at the word or sentence-level. Whereas pitch changes can only be applied to neural voices at the sentence level.

Example

```

<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Guy24kRUS">
    Welcome to <prosody pitch="high">Microsoft Cognitive Services Text-to-Speech API.</prosody>
  </voice>
</speak>

```

Change pitch contour

IMPORTANT

Pitch contour changes aren't supported with neural voices.

Example

```

<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    <prosody contour="(80%,+20%) (90%,+30%)">
      Good morning.
    </prosody>
  </voice>
</speak>

```

say-as element

`say-as` is an optional element that indicates the content type (such as number or date) of the element's text. This provides guidance to the speech synthesis engine about how to pronounce the text.

Syntax

```
<say-as interpret-as="string" format="digit string" detail="string"> <say-as>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
interpret-as	Indicates the content type of element's text. For a list of types, see the table below.	Required
format	Provides additional information about the precise formatting of the element's text for content types that may have ambiguous formats. SSML defines formats for content types that use them (see table below).	Optional
detail	Indicates the level of detail to be spoken. For example, this attribute might request that the speech synthesis engine pronounce punctuation marks. There are no standard values defined for <code>detail</code> .	Optional

The following are the supported content types for the `interpret-as` and `format` attributes. Include the `format` attribute only if `interpret-as` is set to date and time.

INTERPRET-AS	FORMAT	INTERPRETATION
address		<p>The text is spoken as an address. The speech synthesis engine pronounces:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> I'm at <say-as interpret-as="address">150th CT NE, Redmond, WA</say-as> </div> <p>As "I'm at 150th court north east redmond washington."</p>
cardinal, number		<p>The text is spoken as a cardinal number. The speech synthesis engine pronounces:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> There are <say-as interpret-as="cardinal">3</say-as> alternatives </div> <p>As "There are three alternatives."</p>
characters, spell-out		<p>The text is spoken as individual letters (spelled out). The speech synthesis engine pronounces:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <say-as interpret-as="characters">test</say-as> </div> <p>As "T E S T."</p>

INTERPRET-AS	FORMAT	INTERPRETATION
date	dmy, mdy, ymd, ydm, ym, my, md, dm, d, m, y	<p>The text is spoken as a date. The <code>format</code> attribute specifies the date's format (<i>d=day, m=month, and y=year</i>). The speech synthesis engine pronounces:</p> <pre>Today is <say-as interpret-as="date" format="mdy">10-19-2016</say-as></pre> <p>As "Today is October nineteenth two thousand sixteen."</p>
digits, number_digit		<p>The text is spoken as a sequence of individual digits. The speech synthesis engine pronounces:</p> <pre><say-as interpret-as="number_digit">123456789</say-as></pre> <p>As "1 2 3 4 5 6 7 8 9."</p>
fraction		<p>The text is spoken as a fractional number. The speech synthesis engine pronounces:</p> <pre><say-as interpret-as="fraction">3/8</say-as> of an inch</pre> <p>As "three eighths of an inch."</p>
ordinal		<p>The text is spoken as an ordinal number. The speech synthesis engine pronounces:</p> <pre>Select the <say-as interpret-as="ordinal">3rd</say-as> option</pre> <p>As "Select the third option".</p>

INTERPRET-AS	FORMAT	INTERPRETATION
telephone		<p>The text is spoken as a telephone number. The <code>format</code> attribute may contain digits that represent a country code. For example, "1" for the United States or "39" for Italy. The speech synthesis engine may use this information to guide its pronunciation of a phone number. The phone number may also include the country code, and if so, takes precedence over the country code in the <code>format</code>. The speech synthesis engine pronounces:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> The number is <say-as interpret-as="telephone" format="1">(888) 555-1212</say-as> </div> <p>As "My number is area code eight eight eight five five five one two one two."</p>
time	hms12, hms24	<p>The text is spoken as a time. The <code>format</code> attribute specifies whether the time is specified using a 12-hour clock (hms12) or a 24-hour clock (hms24). Use a colon to separate numbers representing hours, minutes, and seconds. The following are valid time examples: 12:35, 1:14:32, 08:15, and 02:50:45. The speech synthesis engine pronounces:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> The train departs at <say-as interpret-as="time" format="hms12">4:00am</say-as> </div> <p>As "The train departs at four A M."</p>

Usage

The `say-as` element may contain only text.

Example

The speech synthesis engine speaks the following example as "Your first request was for one room on October nineteenth twenty ten with early arrival at twelve thirty five P M."

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <voice name="en-US-Jessa24kRUS">
    <p>
      Your <say-as interpret-as="ordinal"> 1st </say-as> request was for <say-as interpret-as="cardinal"> 1
    </say-as> room
    on <say-as interpret-as="date" format="mdy"> 10/19/2010 </say-as>, with early arrival at <say-as interpret-as="time" format="hms12"> 12:35pm </say-as>.
    </p>
  </speak>
```

Add recorded audio

`audio` is an optional element that allows you to insert MP3 audio into an SSML document. The body of the `audio` element may contain plain text or SSML markup that's spoken if the audio file is unavailable or unplayable. Additionally, the `audio` element can contain text and the following elements: `audio`, `break`, `p`, `s`, `phoneme`, `prosody`, `say-as`, and `sub`.

Any audio included in the SSML document must meet these requirements:

- The MP3 must be hosted on an Internet-accessible HTTPS endpoint. HTTPS is required, and the domain hosting the MP3 file must present a valid, trusted SSL certificate.
- The MP3 must be a valid MP3 file (MPEG v2).
- The bit rate must be 48 kbps.
- The sample rate must be 16000 Hz.
- The combined total time for all text and audio files in a single response cannot exceed ninety (90) seconds.
- The MP3 must not contain any customer-specific or other sensitive information.

Syntax

```
<audio src="string"/></audio>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
src	Specifies the location/URL of the audio file.	Required if using the <code>audio</code> element in your SSML document.

Example

```
<speak version="1.0" xmlns="https://www.w3.org/2001/10/synthesis" xml:lang="en-US">
  <p>
    <audio src="https://contoso.com/opinionprompt.wav"/>
    Thanks for offering your opinion. Please begin speaking after the beep.
    <audio src="https://contoso.com/beep.wav">
      Could not play the beep, please voice your opinion now. </audio>
    </p>
  </speak>
```

Add background audio

The `mstts:backgroundaudio` element allows you to add background audio to your SSML documents (or mix an audio file with text-to-speech). With `mstts:backgroundaudio` you can loop an audio file in the background, fade in at the beginning of text-to-speech, and fade out at the end of text-to-speech.

If the background audio provided is shorter than the text-to-speech or the fade out, it will loop. If it is longer than the text-to-speech, it will stop when the fade out has finished.

Only one background audio file is allowed per SSML document. However, you can intersperse `audio` tags within the `voice` element to add additional audio to your SSML document.

Syntax

```
<mstts:backgroundaudio src="string" volume="string" fadein="string" fadeout="string"/>
```

Attributes

ATTRIBUTE	DESCRIPTION	REQUIRED / OPTIONAL
src	Specifies the location/URL of the background audio file.	Required if using background audio in your SSML document.
volume	Specifies the volume of the background audio file. Accepted values: <code>0</code> to <code>100</code> inclusive. The default value is <code>1</code> .	Optional
fadein	Specifies the duration of the background audio fade in in milliseconds. The default value is <code>0</code> , which is the equivalent to no fade in. Accepted values: <code>0</code> to <code>10000</code> inclusive.	Optional
fadeout	Specifies the duration of the background audio fade out in milliseconds. The default value is <code>0</code> , which is the equivalent to no fade out. Accepted values: <code>0</code> to <code>10000</code> inclusive.	Optional

Example

```
<speak version="1.0" xml:lang="en-US" xmlns:mstts="http://www.w3.org/2001/mstts">
  <mstts:backgroundaudio src="https://contoso.com/sample.wav" volume="0.7" fadein="3000" fadeout="4000"/>
  <voice name="Microsoft Server Speech Text to Speech Voice (en-US, Jessa24kRUS)">
    The text provided in this document will be spoken over the background audio.
  </voice>
</speak>
```

Next steps

- [Language support: voices, locales, languages](#)

Get started with Custom Voice

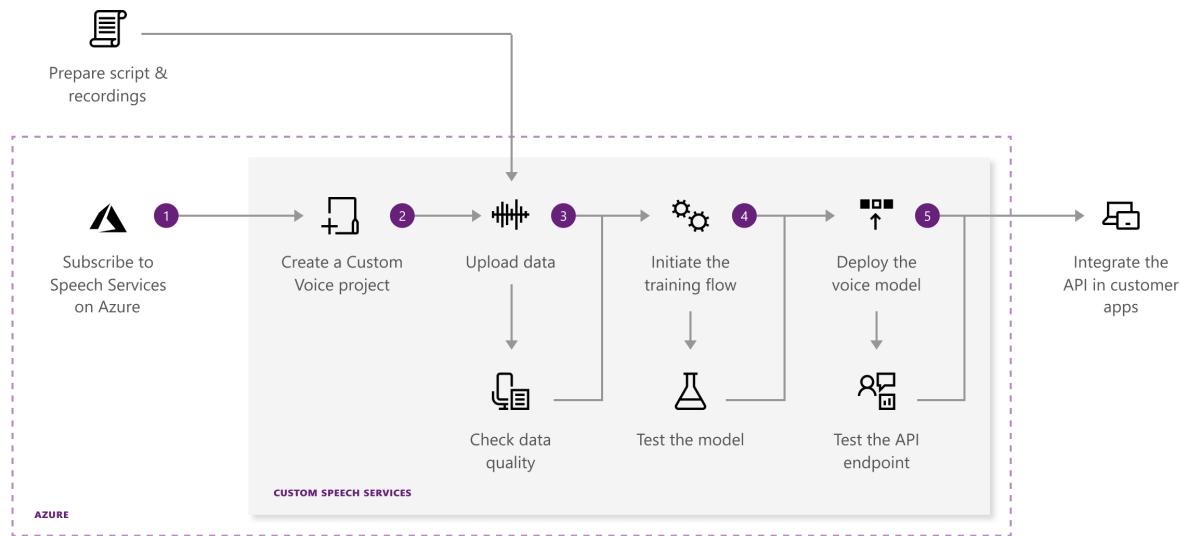
12/4/2019 • 3 minutes to read • [Edit Online](#)

Custom Voice is a set of online tools that allow you to create a recognizable, one-of-a-kind voice for your brand. All it takes to get started are a handful of audio files and the associated transcriptions. Follow the links below to start creating a custom text-to-speech experience.

What's in Custom Voice?

Before starting with Custom Voice, you'll need an Azure account and a Speech service subscription. Once you've created an account, you can prepare your data, train and test your models, evaluate voice quality, and ultimately deploy your custom voice model.

The diagram below highlights the steps to create a custom voice model using the [Custom Voice portal](#). Use the links to learn more.



1. [Subscribe and create a project](#) - Create an Azure account and create a Speech service subscription. This unified subscription gives you access to speech-to-text, text-to-speech, speech translation, and the Custom Voice portal. Then, using your Speech service subscription, create your first Custom Voice project.
2. [Upload data](#) - Upload data (audio and text) using the Custom Voice portal or Custom Voice API. From the portal, you can investigate and evaluate pronunciation scores and signal-to-noise ratios. For more information, see [How to prepare data for Custom Voice](#).
3. [Train your model](#) – Use your data to create a custom text-to-speech voice model. You can train a model in different languages. After training, test your model, and if you're satisfied with the result, you can deploy the model.
4. [Deploy your model](#) - Create a custom endpoint for your text-to-speech voice model, and use it for speech synthesis in your products, tools, and applications.

Custom Neural voices

The neural voice customization capability is currently in public preview, limited to selected customers. Fill out this [application form](#) to get started.

NOTE

As part of Microsoft's commitment to designing responsible AI, our intent is to protect the rights of individuals and society, and foster transparent human-computer interactions. For this reason, Custom Neural Voice is not generally available to all customers. You may gain access to the technology only after your applications are reviewed and you have committed to using it in alignment with our ethics principles. Learn more about our [application gating process](#).

Set up your Azure account

A Speech service subscription is required before you can use the Custom Speech portal to create a custom model. Follow these instructions to create a Speech service subscription in Azure. If you do not have an Azure account, you can sign up for a new one.

Once you've created an Azure account and a Speech service subscription, you'll need to sign in to the Custom Voice portal and connect your subscription.

1. Get your Speech service subscription key from the Azure portal.
2. Sign in to the [Custom Voice portal](#).
3. Select your subscription and create a speech project.
4. If you'd like to switch to another Speech subscription, use the cog icon located in the top navigation.

NOTE

The Custom Voice service does NOT support the 30-day free trial key. You must have a F0 or a S0 key created in Azure before you can use the service.

How to create a project

Content like data, models, tests, and endpoints are organized into **Projects** in the Custom Voice portal. Each project is specific to a country/language and the gender of the voice you want to create. For example, you may create a project for a female voice for your call center's chat bots that use English in the United States (en-US).

To create your first project, select the **Text-to-Speech/Custom Voice** tab, then click **New Project**. Follow the instructions provided by the wizard to create your project. After you've created a project, you will see four tabs: **Data, Training, Testing, and Deployment**. Use the links provided in [Next steps](#) to learn how to use each tab.

Next steps

- [Prepare Custom Voice data](#)
- [Create a Custom Voice](#)
- [Guide: Record your voice samples](#)

Long Audio API (Preview)

12/31/2019 • 2 minutes to read • [Edit Online](#)

The Long Audio API is designed for asynchronous synthesis of long-form text to speech (for example: audio books). This API doesn't return synthesized audio in real-time, instead the expectation is that you will poll for the response(s) and consume the output(s) as they are made available from the service. Unlike the text to speech API that's used by the Speech SDK, the Long Audio API can create synthesized audio longer than 10 minutes, making it ideal for publishers and audio content platforms.

Additional benefits of the Long Audio API:

- Synthesized speech returned by the service uses neural voices, which ensures high-fidelity audio outputs.
- Since real-time responses aren't supported, there's no need to deploy a voice endpoint.

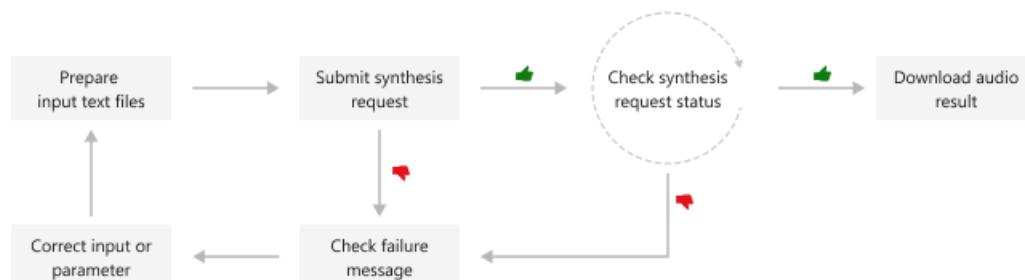
NOTE

The Long Audio API now supports only [Custom Neural Voice](#).

Workflow

Typically, when using the Long Audio API, you'll submit a text file or files to be synthesized, poll for the status, then if the status is successful, you can download the audio output.

This diagram provides a high-level overview of the workflow.



Prepare content for synthesis

When preparing your text file, make sure it:

- Is either plain text (.txt) or SSML text (.txt)
- Is encoded as [UTF-8 with Byte Order Mark \(BOM\)](#)
- Is a single file, not a zip
- Contains more than 400 characters for plain text or 400 [billable characters](#) for SSML text, and less than 10,000 paragraphs
 - For plain text, each paragraph is separated by hitting **Enter/Return** - View [plain text input example](#)
 - For SSML text, each SSML piece is considered a paragraph. SSML pieces shall be separated by different paragraphs - View [SSML text input example](#)

NOTE

For Chinese (Mainland), Chinese (Hong Kong), Chinese (Taiwan), Japanese, and Korean, one word will be counted as two characters.

Submit synthesis requests

After preparing the input content, follow the [long-form audio synthesis quickstart](#) to submit the request. If you have more than one input file, you will need to submit multiple requests. There are some limitations to be aware of:

- Client is allowed to submit up to 5 requests to server per second for each Azure subscription account. If it exceeds the limitation, client will get a 429 error code (too many requests). Please reduce the request amount per second
- Server is allowed to run and queue up to 120 requests for each Azure subscription account. If it exceeds the limitation, server will return a 429 error code (too many requests). Please wait and avoid submitting new request until some requests are completed
- Server will keep up to 20,000 requests for each Azure subscription account. If it exceeds the limitation, please delete some requests before submitting new ones

Audio output formats

We support flexible audio output formats. You can generate audio outputs per paragraph or concatenate the audios into one output by setting the 'concatenateResult' parameter. The following audio output formats are supported by the Long Audio API:

NOTE

The default audio format is riff-16khz-16bit-mono-pcm.

- riff-8khz-16bit-mono-pcm
- riff-16khz-16bit-mono-pcm
- riff-24khz-16bit-mono-pcm
- riff-48khz-16bit-mono-pcm
- audio-16khz-32kbitrate-mono-mp3
- audio-16khz-64kbitrate-mono-mp3
- audio-16khz-128kbitrate-mono-mp3
- audio-24khz-48kbitrate-mono-mp3
- audio-24khz-96kbitrate-mono-mp3
- audio-24khz-160kbitrate-mono-mp3

Quickstarts

We offer quickstarts designed to help you run the Long Audio API successfully. This table includes a list of Long Audio API quickstarts organized by language.

- [Quickstart: Python](#)

Sample code

Sample code for Long Audio API is available on GitHub.

- [Sample code: Python](#)
- [Sample code: C#](#)
- [Sample code: Java](#)

Install and run Speech service containers (Preview)

12/4/2019 • 17 minutes to read • [Edit Online](#)

Containers enable you to run some of the Speech service APIs in your own environment. Containers are great for specific security and data governance requirements. In this article you'll learn how to download, install, and run a Speech container.

Speech containers enable customers to build a speech application architecture that is optimized for both robust cloud capabilities and edge locality. There are four different containers available. The two standard containers are **Speech-to-text** and **Text-to-speech**. The two custom containers are **Custom Speech-to-text** and **Custom Text-to-speech**.

IMPORTANT

All speech containers are currently offered as part of a [Public "Gated" Preview](#). An announcement will be made when speech containers progress to General Availability (GA).

FUNCTION	FEATURES	LATEST
Speech-to-text	Transcribes continuous real-time speech or batch audio recordings into text with intermediate results.	2.0.0
Custom Speech-to-text	Using a custom model from the Custom Speech portal , transcribes continuous real-time speech or batch audio recordings into text with intermediate results.	2.0.0
Text-to-speech	Converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.3.0
Custom Text-to-speech	Using a custom model from the Custom Voice portal , converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.3.0

If you don't have an Azure subscription, create a [free account](#) before you begin.

Prerequisites

The following prerequisites before using Speech containers:

REQUIRED	PURPOSE
----------	---------

REQUIRED	PURPOSE
Docker Engine	<p>You need the Docker Engine installed on a host computer. Docker provides packages that configure the Docker environment on macOS, Windows, and Linux. For a primer on Docker and container basics, see the Docker overview.</p> <p>Docker must be configured to allow the containers to connect with and send billing data to Azure.</p> <p>On Windows, Docker must also be configured to support Linux containers.</p>
Familiarity with Docker	<p>You should have a basic understanding of Docker concepts, like registries, repositories, containers, and container images, as well as knowledge of basic <code>docker</code> commands.</p>
Speech resource	<p>In order to use these containers, you must have:</p> <p>An Azure <i>Speech</i> resource to get the associated API key and endpoint URI. Both values are available on the Azure portal's Speech Overview and Keys pages. They are both required to start the container.</p> <p>{API_KEY}: One of the two available resource keys on the Keys page</p> <p>{ENDPOINT_URI}: The endpoint as provided on the Overview page</p>

Request access to the container registry

Fill out and submit the [Cognitive Services Speech Containers Request form](#) to request access to the container.

The form requests information about you, your company, and the user scenario for which you'll use the container. After you've submitted the form, the Azure Cognitive Services team reviews it to ensure that you meet the criteria for access to the private container registry.

IMPORTANT

You must use an email address that's associated with either a Microsoft Account (MSA) or Azure Active Directory (Azure AD) account in the form.

If your request is approved, you'll receive an email with instructions that describe how to obtain your credentials and access the private container registry.

Use the Docker CLI to authenticate the private container registry

You can authenticate with the private container registry for Cognitive Services Containers in any of several ways, but the recommended method from the command line is to use the [Docker CLI](#).

Use the `docker login` command, as shown in the following example, to log in to `containerpreview.azurecr.io`, the private container registry for Cognitive Services Containers. Replace `<username>` with the user name and `<password>` with the password that's provided in the credentials you received from the Azure Cognitive Services team.

```
docker login containerpreview.azurecr.io -u <username> -p <password>
```

If you've secured your credentials in a text file, you can concatenate the contents of that text file, by using the `cat` command, to the `docker login` command, as shown in the following example. Replace `<passwordFile>` with the path and name of the text file that contains the password and `<username>` with the user name that's provided in your credentials.

```
cat <passwordFile> | docker login containerpreview.azurecr.io -u <username> --password-stdin
```

Gathering required parameters

There are three primary parameters for all Cognitive Services' containers that are required. The end-user license agreement (EULA) must be present with a value of `accept`. Additionally, both an Endpoint URL and API Key are needed.

Endpoint URI {ENDPOINT_URI}

The **Endpoint** URI value is available on the Azure portal *Overview* page of the corresponding Cognitive Service resource. Navigate to the *Overview* page, hover over the Endpoint, and a `Copy to clipboard` icon will appear. Copy and use where needed.

Home > widgets

wIDGETS Cognitive Services

Search (Ctrl+ /)

Overview Copy to clipboard

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

RESOURCE MANAGEMENT

Keys Copy to clipboard

Quick start

Pricing tier

Billing By Subscription

Unavailable setting Delete

Resource group (change)
widgets-resource-group

Status
Active

Location
North Central US

Subscription (change)
widgets-subscription

Subscription ID
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Tags (change)
Click here to add tags

API type
<API Type>

Pricing tier
Standard

Endpoint
<https://widgets.cognitiveservices.azure.com/api/example-endpoint>

Manage keys
Show access keys ...

Keys {API_KEY}

This key is used to start the container, and is available on the Azure portal's Keys page of the corresponding Cognitive Service resource. Navigate to the *Keys* page, and click on the `Copy to clipboard` icon.

Home > widgets - Keys

wIDGETS - Keys Cognitive Services

Search (Ctrl+ /)

Regenerate Key1 Regenerate Key2

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

RESOURCE MANAGEMENT

Keys Copy to clipboard

Quick start

Pricing tier

Billing By Subscription

NAME
widgets

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

KEY 1
key 1 value Copy to clipboard

KEY 2
key 2 value Copy to clipboard

IMPORTANT

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely, for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

The host computer

The host is a x64-based computer that runs the Docker container. It can be a computer on your premises or a Docker hosting service in Azure, such as:

- [Azure Kubernetes Service](#).
- [Azure Container Instances](#).
- A [Kubernetes](#) cluster deployed to [Azure Stack](#). For more information, see [Deploy Kubernetes to Azure Stack](#).

Advanced Vector Extension support

The **host** is the computer that runs the docker container. The host *must support* [Advanced Vector Extensions](#) (AVX2). You can check for AVX2 support on Linux hosts with the following command:

```
grep -q avx2 /proc/cpuinfo && echo AVX2 supported || echo No AVX2 support detected
```

WARNING

The host computer is *required* to support AVX2. The container *will not* function correctly without AVX2 support.

Container requirements and recommendations

The following table describes the minimum and recommended allocation of resources for each Speech container.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

CONTAINER	MINIMUM	RECOMMENDED
Speech-to-text	2 core, 2-GB memory	4 core, 4-GB memory

- Each core must be at least 2.6 gigahertz (GHz) or faster.

Core and memory correspond to the `--cpus` and `--memory` settings, which are used as part of the `docker run` command.

NOTE

The minimum and recommended are based off of Docker limits, *not* the host machine resources. For example, speech-to-text containers memory map portions of a large language model, and it is *recommended* that the entire file fits in memory, which is an additional 4-6 GB. Also, the first run of either container may take longer, since models are being paged into memory.

Get the container image with `docker pull`

Container images for Speech are available in the following Container Registry.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

CONTAINER	REPOSITORY
Speech-to-text	containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text:latest

TIP

You can use the [docker images](#) command to list your downloaded container images. For example, the following command lists the ID, repository, and tag of each downloaded container image, formatted as a table:

```
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
IMAGE ID          REPOSITORY          TAG
<image-id>      <repository-path/name>  <tag-name>
```

Docker pull for the Speech containers

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

Docker pull for the Speech-to-text container

Use the [docker pull](#) command to download a container image from Container Preview registry.

```
docker pull containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text:latest
```

IMPORTANT

The `latest` tag pulls the `en-us` locale. For additional locales see [Speech-to-text locales](#).

Speech-to-text locales

All tags, except for `latest` are in the following format and are case-sensitive:

```
<major>.<minor>.<patch>-<platform>-<locale>-<prerelease>
```

The following tag is an example of the format:

```
2.0.0-amd64-en-us-preview
```

For all of the supported locales of the **speech-to-text** container, please see [Speech-to-text image tags](#).

How to use the container

Once the container is on the [host computer](#), use the following process to work with the container.

1. [Run the container](#), with the required billing settings. More [examples](#) of the `docker run` command are available.

2. Query the container's prediction endpoint

Run the container with `docker run`

Use the `docker run` command to run the container. Refer to [gathering required parameters](#) for details on how to get the `{Endpoint_URI}` and `{API_Key}` values. Additional [examples](#) of the `docker run` command are also available.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

To run the *Speech-to-text* container, execute the following `docker run` command.

```
docker run --rm -it -p 5000:5000 --memory 4g --cpus 4 \
containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

This command:

- Runs a *Speech-to-text* container from the container image.
- Allocates 4 CPU cores and 4 gigabytes (GB) of memory.
- Exposes TCP port 5000 and allocates a pseudo-TTY for the container.
- Automatically removes the container after it exits. The container image is still available on the host computer.

IMPORTANT

The `Eula`, `Billing`, and `ApiKey` options must be specified to run the container; otherwise, the container won't start. For more information, see [Billing](#).

Query the container's prediction endpoint

CONTAINERS	SDK HOST URL	PROTOCOL
Speech-to-text and Custom Speech-to-text	<code>ws://localhost:5000</code>	WS
Text-to-speech and Custom Text-to-speech	<code>http://localhost:5000</code>	HTTP

For more information on using WSS and HTTPS protocols, see [container security](#).

Speech-to-text or Custom Speech-to-text

The container provides websocket-based query endpoint APIs, that are accessed through the [Speech SDK](#). By default, the Speech SDK uses online speech services. To use the container, you need to change the initialization method.

TIP

When using the Speech SDK with containers, you do not need to provide the Azure Speech resource [subscription key](#) or an [authentication bearer token](#).

See the examples below.

- C#
- Python

Change from using this Azure-cloud initialization call:

```
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

to this call using the container **host**:

```
var config = SpeechConfig.FromHost(  
    new Uri("ws://localhost:5000"));
```

Text-to-speech or Custom Text-to-speech

The container provides [REST-based endpoint APIs](#). There are many [sample source code projects](#) for platform, framework, and language variations available.

With the *Standard Text-to-speech* container, you should rely on the locale and voice of the image tag you downloaded. For example, if you downloaded the `latest` tag the default locale is `en-US` and the `JessaRUS` voice. The `{VOICE_NAME}` argument would then be `en-US-JessaRUS`. See the example SSML below:

```
<speak version="1.0" xml:lang="en-US">  
  <voice name="en-US-JessaRUS">  
    This text will get converted into synthesized speech.  
  </voice>  
</speak>
```

However, for *Custom Text-to-speech* you'll need to obtain the **Voice / model** from the [custom voice portal](#). The custom model name is synonymous with the voice name. Navigate to the **Training** page, and copy the **Voice / model** to use as the `{VOICE_NAME}` argument.

Text input ↑	Voice / model ↑	Created ↓	Status ↑↓	Audio
Hi. this is my custom voice.	custom-voice-model	10/15/2019 7:29 AM	Succeeded	[Speaker icon]

See the example SSML below:

```
<speak version="1.0" xml:lang="en-US">  
  <voice name="custom-voice-model">  
    This text will get converted into synthesized speech.  
  </voice>  
</speak>
```

Let's construct an HTTP POST request, providing a few headers and a data payload. Replace the `{VOICE_NAME}`

placeholder with your own value.

```
curl -s -v -X POST http://localhost:5000/speech/synthesize/cognitiveservices/v1 \
-H 'Accept: audio/*' \
-H 'Content-Type: application/ssml+xml' \
-H 'X-Microsoft-OutputFormat: riff-16khz-16bit-mono-pcm' \
-d '<speak version="1.0" xml:lang="en-US"><voice name="{VOICE_NAME}">This is a test, only a test.</voice>
</speak>'
```

This command:

- Constructs an HTTP POST request for the `speech/synthesize/cognitiveservices/v1` endpoint.
- Specifies an `Accept` header of `audio/*`
- Specifies a `Content-Type` header of `application/ssml+xml`, for more information, see [request body](#).
- Specifies a `X-Microsoft-OutputFormat` header of `riff-16khz-16bit-mono-pcm`, for more options see [audio output](#).
- Sends the [Speech Synthesis Markup Language \(SSML\)](#) request given the `{VOICE_NAME}` to the endpoint.

Run multiple containers on the same host

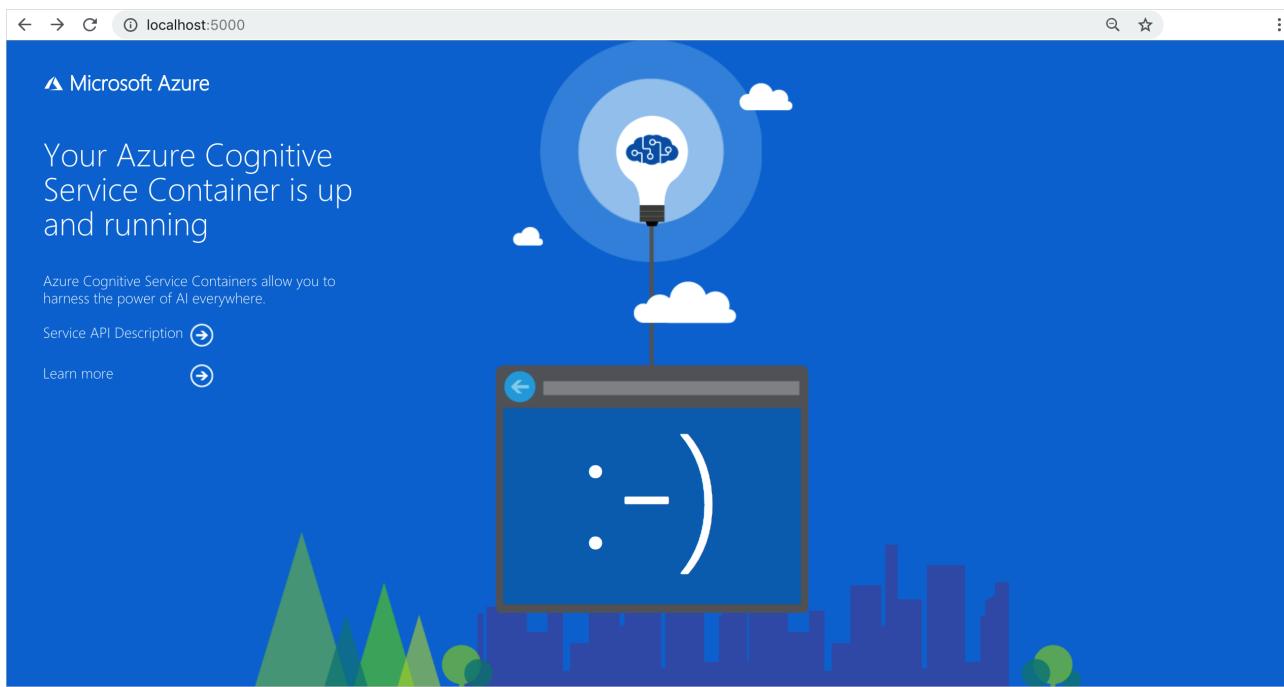
If you intend to run multiple containers with exposed ports, make sure to run each container with a different exposed port. For example, run the first container on port 5000 and the second container on port 5001.

You can have this container and a different Azure Cognitive Services container running on the HOST together. You also can have multiple containers of the same Cognitive Services container running.

Validate that a container is running

There are several ways to validate that the container is running. Locate the *External IP* address and exposed port of the container in question, and open your favorite web browser. Use the various request URLs below to validate the container is running. The example request URLs listed below are `http://localhost:5000`, but your specific container may vary. Keep in mind that you're to rely on your container's *External IP* address and exposed port.

REQUEST URL	PURPOSE
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/status</code>	Requested with an HTTP GET, to validate that the container is running without causing an endpoint query. This request can be used for Kubernetes liveness and readiness probes .
<code>http://localhost:5000/swagger</code>	The container provides a full set of documentation for the endpoints and a Try it out feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.



Stop the container

To shut down the container, in the command-line environment where the container is running, select **Ctrl+C**.

Troubleshooting

When starting or running the container, you may experience issues. Use an output **mount** and enable logging. Doing so will allow the container to generate log files that are helpful when troubleshooting issues.

TIP

For more troubleshooting information and guidance, see [Cognitive Services containers frequently asked questions \(FAQ\)](#).

Billing

The Speech containers send billing information to Azure, using a *Speech* resource on your Azure account.

Queries to the container are billed at the pricing tier of the Azure resource that's used for the `<ApiKey>`.

Azure Cognitive Services containers aren't licensed to run without being connected to the billing endpoint for metering. You must enable the containers to communicate billing information with the billing endpoint at all times. Cognitive Services containers don't send customer data, such as the image or text that's being analyzed, to Microsoft.

Connect to Azure

The container needs the billing argument values to run. These values allow the container to connect to the billing endpoint. The container reports usage about every 10 to 15 minutes. If the container doesn't connect to Azure within the allowed time window, the container continues to run but doesn't serve queries until the billing endpoint is restored. The connection is attempted 10 times at the same time interval of 10 to 15 minutes. If it can't connect to the billing endpoint within the 10 tries, the container stops running.

Billing arguments

For the `docker run` command to start the container, all three of the following options must be specified with valid values:

OPTION	DESCRIPTION
ApiKey	The API key of the Cognitive Services resource that's used to track billing information. The value of this option must be set to an API key for the provisioned resource that's specified in <code>Billing</code> .
Billing	The endpoint of the Cognitive Services resource that's used to track billing information. The value of this option must be set to the endpoint URI of a provisioned Azure resource.
Eula	Indicates that you accepted the license for the container. The value of this option must be set to <code>accept</code> .

For more information about these options, see [Configure containers](#).

Blog posts

- [Running Cognitive Services Containers](#)
- [Azure Cognitive Services](#)

Developer samples

Developer samples are available at our [GitHub repository](#).

View webinar

Join the [webinar](#) to learn about:

- How to deploy Cognitive Services to any machine using Docker
- How to deploy Cognitive Services to AKS

Summary

In this article, you learned concepts and workflow for downloading, installing, and running Speech containers. In summary:

- Speech provides four Linux containers for Docker, encapsulating various capabilities:
 - *Speech-to-text*
 - *Custom Speech-to-text*
 - *Text-to-speech*
 - *Custom Text-to-speech*
- Container images are downloaded from the container registry in Azure.
- Container images run in Docker.
- You can use either the REST API or SDK to call operations in Speech containers by specifying the host URL of the container.
- You're required to provide billing information when instantiating a container.

IMPORTANT

Cognitive Services containers are not licensed to run without being connected to Azure for metering. Customers need to enable the containers to communicate billing information with the metering service at all times. Cognitive Services containers do not send customer data (e.g., the image or text that is being analyzed) to Microsoft.

Next steps

- Review [configure containers](#) for configuration settings
- Learn how to [use Speech service containers with Kubernetes and Helm](#)
- Use more [Cognitive Services containers](#)

Release notes

12/16/2019 • 14 minutes to read • [Edit Online](#)

Speech SDK 1.8.0: 2019-November release

New Features

- Added a `FromHost()` API, to ease use with on-prem containers and sovereign clouds.
- Added Automatic Source Language Detection for Speech Recognition (in Java and C++)
- Added `SourceLanguageConfig` object for Speech Recognition, used to specify expected source languages (in Java and C++)
- Added `KeywordRecognizer` support on Windows (UWP), Android and iOS through the Nuget and Unity packages
- Added Remote Conversation Java API to do Conversation Transcription in asynchronous batches.

Breaking changes

- Conversation Transcriber functionalities moved under namespace
`Microsoft.CognitiveServices.Speech.Transcription`.
- Part of the Conversation Transcriber methods are moved to new `Conversation` class.
- Dropped support for 32-bit (ARMv7 and x86) iOS

Bug fixes

- Fix for crash if local `KeywordRecognizer` is used without a valid speech service subscription key

Samples

- Xamarin sample for `KeywordRecognizer`
- Unity sample for `KeywordRecognizer`
- C++ and Java samples for Automatic Source Language Detection.

Speech SDK 1.7.0: 2019-September release

New Features

- Added beta support for Xamarin on Universal Windows Platform (UWP), Android, and iOS
- Added iOS support for Unity
- Added `Compressed` input support for ALaw, Mulaw, FLAC on Android, iOS and Linux
- Added `SendMessageAsync` in `Connection` class for sending a message to service
- Added `SetMessageProperty` in `Connection` class for setting property of a message
- TTS added bindings for Java (Jre and Android), Python, Swift, and Objective-C
- TTS added playback support for macOS, iOS, and Android.
- Added "word boundary" information for TTS.

Bug fixes

- Fixed IL2CPP build issue on Unity 2019 for Android
- Fixed issue with malformed headers in wav file input being processed incorrectly
- Fixed issue with UUIDs not being unique in some connection properties

- Fixed a few warnings about nullability specifiers in the Swift bindings (might require small code changes)
- Fixed a bug that caused websocket connections to be closed ungracefully under network load
- Fixed an issue on Android that sometimes results in duplicate impression IDs used by `DialogServiceConnector`
- Improvements to the stability of connections across multi-turn interactions and the reporting of failures (via `Canceled` events) when they occur with `DialogServiceConnector`
- `DialogServiceConnector` session starts will now properly provide events, including when calling `ListenOnceAsync()` during an active `StartKeywordRecognitionAsync()`
- Addressed a crash associated with `DialogServiceConnector` activities being received

Samples

- Quickstart for Xamarin
- Updated CPP Quickstart with Linux ARM64 information
- Updated Unity quickstart with iOS information

Speech SDK 1.6.0: 2019-June release

Samples

- Quickstart samples for Text To Speech on UWP and Unity
- Quickstart sample for Swift on iOS
- Unity samples for Speech & Intent Recognition and Translation
- Updated quickstart samples for `DialogServiceConnector`

Improvements / Changes

- Dialog namespace:
 - `SpeechBotConnector` has been renamed to `DialogServiceConnector`
 - `BotConfig` has been renamed to `DialogServiceConfig`
 - `BotConfig::FromChannelSecret()` has been remapped to `DialogServiceConfig::FromBotSecret()`
 - All existing Direct Line Speech clients continue to be supported after the rename
- Update TTS REST adapter to support proxy, persistent connection
- Improve error message when an invalid region is passed
- Swift/Objective-C:
 - Improved error reporting: Methods that can result in an error are now present in two versions: One that exposes an `NSError` object for error handling, and one that raises an exception. The former are exposed to Swift. This change requires adaptations to existing Swift code.
 - Improved event handling

Bug fixes

- Fix for TTS: where `SpeakTextAsync` future returned without waiting until audio has completed rendering
- Fix for marshaling strings in C# to enable full language support
- Fix for .NET core app problem to load core library with net461 target framework in samples
- Fix for occasional issues to deploy native libraries to the output folder in samples
- Fix for web socket closing reliably
- Fix for possible crash while opening a connection under very heavy load on Linux
- Fix for missing metadata in the framework bundle for macOS
- Fix for problems with `pip install --user` on Windows

Speech SDK 1.5.1

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see [here](#).
- Conversation transcription functionality is available through the SDK. See [here](#).
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `outputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the [text-to-speech overview](#).
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details [here](#)).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C

support for macOS is currently in beta.

- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.
- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details [here](#)).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing [audio input device selection](#). This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the [GitHub sample repository](#). This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our [Unity quickstart](#).
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our [sample repository](#):

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `connection` object in iOS.
- Additional Java samples for translation with audio output.

- New sample for use of the [Batch Transcription REST API](#).

Improvements / Changes

- Python
 - Improved parameter verification and error messages in `SpeechConfig`.
 - Add support for the `Connection` object.
 - Support for 32-bit Python (x86) on Windows.
 - The Speech SDK for Python is out of beta.
- iOS
 - The SDK is now built against the iOS SDK version 12.1.
 - The SDK now supports iOS versions 9.2 and later.
 - Improve reference documentation and fix several property names.
- JavaScript
 - Add support for the `Connection` object.
 - Add type definition files for bundled JavaScript
 - Initial support and implementation for phrase hints.
 - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at `turn.end`, not at `speech.end`.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
 - The Beta version of Python support (3.5 and above) is available with this release. For more information, see [here](#)](quickstart-python.md).
- JavaScript

- The Speech SDK for JavaScript has been open-sourced. The source code is available on [GitHub](#).
- We now support Node.js, more info can be found [here](#).
- The length restriction for audio sessions has been removed, reconnection will happen automatically under the cover.
- `Connection` object
 - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
 - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; `RecognizedIntent` was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the [sample repository](#).

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.

- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in `AddIntent` in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the [sample repository](#).

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our [sample repository](#), a new sample for JavaScript was added.

Cognitive Services Speech SDK 1.0.0: 2018-September release

New features

- Support for Objective-C on iOS. Check out our [Objective-C quickstart for iOS](#).
- Support for JavaScript in browser. Check out our [JavaScript quickstart](#).

Breaking changes

- With this release, a number of breaking changes are introduced. Check [this page](#) for details.

Cognitive Services Speech SDK 0.6.0: 2018-August release

New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the [UWP quickstart](#).
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the [Java Runtime](#)

[Environment quickstart](#).

Functional change

- Expose additional error detail information on connection errors.

Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.
- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

Cognitive Services Speech SDK 0.5.0: 2018-July release

New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the [Android quickstart](#).
- Support .NET Standard 2.0 on Windows. Check out the [.NET Core quickstart](#).
- Experimental: Support UWP on Windows (version 1709 or later).
 - Check out the [UWP quickstart](#).
 - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support `AuthorizationToken` for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- `SpeechOutputFormat` in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
 - In Java, the `read` method now returns `long` instead of `int`.
 - In C#, the `Read` method now returns `uint` instead of `int`.
 - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a [regions](#) page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- `AudioInputStream`

A recognizer now can consume a stream as the audio source. For more information, see the related [how-to guide](#).

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- Fired a `NoMatch` result when the response from service is timed out.
- The media foundation libraries on Windows are delay loaded. This library is required for microphone input only.
- The upload speed for audio data is limited to about twice the original audio speed.
- On Windows, C# .NET assemblies now are strong named.
- Documentation fix: `Region` is required information to create a recognizer.

More samples have been added and are constantly being updated. For the latest set of samples, see the [Speech SDK samples GitHub repository](#).

Cognitive Services Speech SDK 0.2.12733: 2018-May release

This release is the first public preview release of the Cognitive Services Speech SDK.

Text-to-speech REST API

12/10/2019 • 9 minutes to read • [Edit Online](#)

The Speech service allows you to [convert text into synthesized speech](#) and [get a list of supported voices](#) for a region using a set of REST APIs. Each available endpoint is associated with a region. A subscription key for the endpoint/region you plan to use is required.

The text-to-speech REST API supports neural and standard text-to-speech voices, each of which supports a specific language and dialect, identified by locale.

- For a complete list of voices, see [language support](#).
- For information about regional availability, see [regions](#).

IMPORTANT

Costs vary for standard, custom, and neural voices. For more information, see [Pricing](#).

Before using this API, understand:

- The text-to-speech REST API requires an Authorization header. This means that you need to complete a token exchange to access the service. For more information, see [Authentication](#).

Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each service:

SUPPORTED AUTHORIZATION HEADERS	SPEECH-TO-TEXT	TEXT-TO-SPEECH
Ocp-Apim-Subscription-Key	Yes	No
Authorization: Bearer	Yes	Yes

When using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your subscription key. For example:

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your subscription key for an access token that's valid for 10 minutes. In the next few sections you'll learn how to get a token, and use a token.

How to get an access token

To get an access token, you'll need to make a request to the `issueToken` endpoint using the `Ocp-Apim-Subscription-Key` and your subscription key.

These regions and endpoints are supported:

REGION	TOKEN SERVICE ENDPOINT
Australia East	https://australiaeast.api.cognitive.microsoft.com/sts/v1.0/issueToken
Canada Central	https://canadacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken
Central US	https://centralus.api.cognitive.microsoft.com/sts/v1.0/issueToken
East Asia	https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken

REGION	TOKEN SERVICE ENDPOINT
East US	https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken
East US 2	https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken
France Central	https://francecentral.api.cognitive.microsoft.com/sts/v1.0/issueToken
India Central	https://centralindia.api.cognitive.microsoft.com/sts/v1.0/issueToken
Japan East	https://japaneast.api.cognitive.microsoft.com/sts/v1.0/issueToken
Korea Central	https://koreacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken
North Central US	https://northcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken
North Europe	https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken
South Central US	https://southcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken
Southeast Asia	https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken
UK South	https://uksouth.api.cognitive.microsoft.com/sts/v1.0/issueToken
West Europe	https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken
West US	https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
West US 2	https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken

Use these samples to create your access token request.

HTTP sample

This example is a simple HTTP request to get a token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.api.cognitive.microsoft.com
Content-type: application/x-www-form-urlencoded
Content-Length: 0
```

The body of the response contains the access token in JSON Web Token (JWT) format.

PowerShell sample

This example is a simple PowerShell script to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
$FetchTokenHeader = @{
    'Content-type'='application/x-www-form-urlencoded';
    'Content-Length'= '0';
    'Ocp-Apim-Subscription-Key' = 'YOUR_SUBSCRIPTION_KEY'
}

$OAuthToken = Invoke-RestMethod -Method POST -Uri https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
-Headers $FetchTokenHeader

# show the token received
$OAuthToken
```

cURL sample

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). This cURL command illustrates how to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
curl -v -X POST
"https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
-H "Content-type: application/x-www-form-urlencoded" \
-H "Content-Length: 0" \
-H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

C# sample

This C# class illustrates how to get an access token. Pass your Speech Service subscription key when you instantiate the class. If your subscription isn't in the West US region, change the value of `FetchTokenUri` to match the region for your subscription.

```
public class Authentication
{
    public static readonly string FetchTokenUri =
        "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
    private string subscriptionKey;
    private string token;

    public Authentication(string subscriptionKey)
    {
        this.subscriptionKey = subscriptionKey;
        this.token = FetchTokenAsync(FetchTokenUri, subscriptionKey).Result;
    }

    public string GetAccessToken()
    {
        return this.token;
    }

    private async Task<string> FetchTokenAsync(string fetchUri, string subscriptionKey)
    {
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
            UriBuilder uriBuilder = new UriBuilder(fetchUri);

            var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
            Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
            return await result.Content.ReadAsStringAsync();
        }
    }
}
```

Python sample

```
# Request module must be installed.
# Run pip install requests if necessary.
import requests

subscription_key = 'REPLACE_WITH_YOUR_KEY'

def get_token(subscription_key):
    fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key
    }
    response = requests.post(fetch_token_url, headers=headers)
    access_token = str(response.text)
    print(access_token)
```

How to use an access token

The access token should be sent to the service as the `Authorization: Bearer <TOKEN>` header. Each access token is valid for 10 minutes. You can get a new token at any time, however, to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the text-to-speech REST API:

```
POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.stt.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

// Message body here...
```

Get a list of voices

The `voices/list` endpoint allows you to get a full list of voices for a specific region/endpoint.

Regions and endpoints

REGION	ENDPOINT
Australia East	https://australiaeast.tts.speech.microsoft.com/cognitiveservices/voices/list
Brazil South	https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/voices/list
Canada Central	https://canadacentral.tts.speech.microsoft.com/cognitiveservices/voices/list
Central US	https://centralus.tts.speech.microsoft.com/cognitiveservices/voices/list
East Asia	https://eastasia.tts.speech.microsoft.com/cognitiveservices/voices/list
East US	https://eastus.tts.speech.microsoft.com/cognitiveservices/voices/list
East US 2	https://eastus2.tts.speech.microsoft.com/cognitiveservices/voices/list
France Central	https://francecentral.tts.speech.microsoft.com/cognitiveservices/voices/list
India Central	https://centralindia.tts.speech.microsoft.com/cognitiveservices/voices/list
Japan East	https://japaneast.tts.speech.microsoft.com/cognitiveservices/voices/list
Korea Central	https://koreacentral.tts.speech.microsoft.com/cognitiveservices/voices/list
North Central US	https://northcentralus.tts.speech.microsoft.com/cognitiveservices/voices/list
North Europe	https://northeurope.tts.speech.microsoft.com/cognitiveservices/voices/list
South Central US	https://southcentralus.tts.speech.microsoft.com/cognitiveservices/voices/list
Southeast Asia	https://southeastasia.tts.speech.microsoft.com/cognitiveservices/voices/list
UK South	https://uksouth.tts.speech.microsoft.com/cognitiveservices/voices/list
West Europe	https://westeurope.tts.speech.microsoft.com/cognitiveservices/voices/list
West US	https://westus.tts.speech.microsoft.com/cognitiveservices/voices/list
West US 2	https://westus2.tts.speech.microsoft.com/cognitiveservices/voices/list

Request headers

This table lists required and optional headers for text-to-speech requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
<code>Authorization</code>	An authorization token preceded by the word <code>Bearer</code> . For more information, see Authentication .	Required

Request body

A body isn't required for `GET` requests to this endpoint.

Sample request

This request only requires an authorization header.

```
GET /cognitiveservices/voices/list HTTP/1.1
Host: westus.tts.speech.microsoft.com
Authorization: Bearer [Base64 access_token]
```

Sample response

This response has been truncated to illustrate the structure of a response.

NOTE

Voice availability varies by region/endpoint.

```
[
  {
    "Name": "Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)",
    "ShortName": "ar-EG-Hoda",
    "Gender": "Female",
    "Locale": "ar-EG"
  },
  {
    "Name": "Microsoft Server Speech Text to Speech Voice (ar-SA, Naayf)",
    "ShortName": "ar-SA-Naayf",
    "Gender": "Male",
    "Locale": "ar-SA"
  },
  {
    "Name": "Microsoft Server Speech Text to Speech Voice (bg-BG, Ivan)",
    "ShortName": "bg-BG-Ivan",
    "Gender": "Male",
    "Locale": "bg-BG"
  },
  {
    "Name": "Microsoft Server Speech Text to Speech Voice (ca-ES, HerenaRUS)",
    "ShortName": "ca-ES-HerenaRUS",
    "Gender": "Female",
    "Locale": "ca-ES"
  },
  {
    "Name": "Microsoft Server Speech Text to Speech Voice (cs-CZ, Jakub)",
    "ShortName": "cs-CZ-Jakub",
    "Gender": "Male",
    "Locale": "cs-CZ"
  },
  ...
]
```

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
200	OK	The request was successful.
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
429	Too Many Requests	You have exceeded the quota or rate of requests allowed for your subscription.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

Convert text-to-speech

The `v1` endpoint allows you to convert text-to-speech using [Speech Synthesis Markup Language \(SSML\)](#).

Regions and endpoints

These regions are supported for text-to-speech using the REST API. Make sure that you select the endpoint that matches your subscription region.

Standard and neural voices

Use this table to determine availability of standard and neural voices by region/endpoint:

REGION	ENDPOINT	STANDARD VOICES	NEURAL VOICES
Australia East	https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
Canada Central	https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
Central US	https://centralus.tts.speech.microsoft.com/cognitiveservices/v1	No	No
East Asia	https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
East US	https://eastus.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
East US 2	https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
France Central	https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
India Central	https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
Japan East	https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
Korea Central	https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
North Central US	https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
North Europe	https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
South Central US	https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
Southeast Asia	https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes

REGION	ENDPOINT	STANDARD VOICES	NEURAL VOICES
UK South	https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
West Europe	https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes
West US	https://westus.tts.speech.microsoft.com/cognitiveservices/v1	Yes	No
West US 2	https://westus2.tts.speech.microsoft.com/cognitiveservices/v1	Yes	Yes

Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Canada Central	https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Central US	https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
East Asia	https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
East US	https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
East US 2	https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
France Central	https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
India Central	https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Japan East	https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Korea Central	https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
North Central US	https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
North Europe	https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
South Central US	https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
Southeast Asia	https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
UK South	https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West Europe	https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}

REGION	ENDPOINT
West US	https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}
West US 2	https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}

Request headers

This table lists required and optional headers for text-to-speech requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
<code>Authorization</code>	An authorization token preceded by the word <code>Bearer</code> . For more information, see Authentication .	Required
<code>Content-Type</code>	Specifies the content type for the provided text. Accepted value: <code>application/ssml+xml</code>	Required
<code>X-Microsoft-OutputFormat</code>	Specifies the audio output format. For a complete list of accepted values, see audio outputs .	Required
<code>User-Agent</code>	The application name. The value provided must be less than 255 characters.	Required

Audio outputs

This is a list of supported audio formats that are sent in each request as the `x-Microsoft-OutputFormat` header. Each incorporates a bitrate and encoding type. The Speech service supports 24 kHz, 16 kHz, and 8 kHz audio outputs.

<code>raw-16khz-16bit-mono-pcm</code>	<code>raw-8khz-8bit-mono-mulaw</code>
<code>riff-8khz-8bit-mono-alaw</code>	<code>riff-8khz-8bit-mono-mulaw</code>
<code>riff-16khz-16bit-mono-pcm</code>	<code>audio-16khz-128kbitrate-mono-mp3</code>
<code>audio-16khz-64kbitrate-mono-mp3</code>	<code>audio-16khz-32kbitrate-mono-mp3</code>
<code>raw-24khz-16bit-mono-pcm</code>	<code>riff-24khz-16bit-mono-pcm</code>
<code>audio-24khz-160kbitrate-mono-mp3</code>	<code>audio-24khz-96kbitrate-mono-mp3</code>
<code>audio-24khz-48kbitrate-mono-mp3</code>	

NOTE

If your selected voice and output format have different bit rates, the audio is resampled as necessary. However, 24 kHz voices do not support `audio-16khz-16kbps-mono-siren` and `riff-16khz-16kbps-mono-siren` output formats.

Request body

The body of each `POST` request is sent as [Speech Synthesis Markup Language \(SSML\)](#). SSML allows you to choose the voice and language of the synthesized speech returned by the text-to-speech service. For a complete list of supported voices, see [language support](#).

NOTE

If using a custom voice, the body of a request can be sent as plain text (ASCII or UTF-8).

Sample request

This HTTP request uses SSML to specify the voice and language. The body cannot exceed 1,000 characters.

```
POST /cognitiveservices/v1 HTTP/1.1

X-Microsoft-OutputFormat: raw-16khz-16bit-mono-pcm
Content-Type: application/ssml+xml
Host: westus.tts.speech.microsoft.com
Content-Length: 225
Authorization: Bearer [Base64 access_token]

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Female'
name='en-US-JessaRUS'>
    Microsoft Speech Service Text-to-Speech API
</voice></speak>
```

See our quickstarts for language-specific examples:

- [.NET Core, C#](#)
- [Python](#)
- [Node.js](#)

HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
200	OK	The request was successful; the response body is an audio file.
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
413	Request Entity Too Large	The SSML input is longer than 1024 characters.
415	Unsupported Media Type	It's possible that the wrong Content-Type was provided. Content-Type should be set to application/ssml+xml .
429	Too Many Requests	You have exceeded the quota or rate of requests allowed for your subscription.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

If the HTTP status is 200 OK , the body of the response contains an audio file in the requested format. This file can be played as it's transferred, saved to a buffer, or saved to a file.

Next steps

- [Get your Speech trial subscription](#)
- [Customize acoustic models](#)

- Customize language models

Text to Speech frequently asked questions

11/14/2019 • 3 minutes to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, check out [other support options](#).

General

Q: What is the difference between a standard voice model and a custom voice model?

A: The standard voice model (also called a *voice font*) has been trained by using Microsoft-owned data and is already deployed in the cloud. You can use a custom voice model either to adapt an average model and transfer the timbre and expression of the speaker's voice style or train a full, new model based on the training data prepared by the user. Today, more and more customers want a one-of-a-kind, branded voice for their bots. The custom voice-building platform is the right choice for that option.

Q: Where do I start if I want to use a standard voice model?

A: More than 80 standard voice models in over 45 languages are available through HTTP requests. First, get a [subscription key](#). To make REST calls to the predeployed voice models, see the [REST API](#).

Q: If I want to use a customized voice model, is the API the same as the one that's used for standard voices?

A: When a custom voice model is created and deployed, you get a unique endpoint for your model. To use the voice to speak in your apps, you must specify the endpoint in your HTTP requests. The same functionality that's available in the REST API for the Text to Speech service is available for your custom endpoint. Learn how to [create and use your custom endpoint](#).

Q: Do I need to prepare the training data to create custom voice models on my own?

A: Yes, you must prepare the training data yourself for a custom voice model.

A collection of speech data is required to create a customized voice model. This collection consists of a set of audio files of speech recordings and a text file of the transcription of each audio file. The result of your digital voice relies heavily on the quality of your training data. To produce a good text-to-speech voice, it's important that the recordings are made in a quiet room with a high-quality, standing microphone. A consistent volume, speaking rate, and speaking pitch, and even consistency in expressive mannerisms of speech are essential for building a great digital voice. We highly recommend recording the voices in a recording studio.

Currently, we don't provide online recording support or have any recording studio recommendations. For the format requirement, see [how to prepare recordings and transcripts](#).

Q: What scripts should I use to record the speech data for custom voice training?

A: We don't limit the scripts for voice recording. You can use your own scripts to record the speech. Just ensure that you have sufficient phonetic coverage in your speech data. To train a custom voice, you can start with a small volume of speech data, which might be 50 different sentences (about 3-5 minutes of speech). The more data you provide, the more natural your voice will be. You can start to train a full voice font when you provide recordings of more than 2,000 sentences (about 3-4 hours of speech). To get a high-quality, full voice, you need to prepare recordings of more than 6,000 sentences (about 8-10 hours of speech).

We provide additional services to help you prepare scripts for recording. Contact [Custom Voice customer support](#) for inquiries.

Q: What if I need higher concurrency than the default value or what is offered in the portal?

A: You can scale up your model in increments of 20 concurrent requests. Contact [Custom Voice customer support](#) for inquiries about higher scaling.

Q: Can I download my model and run it locally?

A: Models can't be downloaded and executed locally.

Q: Are my requests throttled?

A: The REST API limits requests to 25 per 5 seconds. Details can be found in our pages for [Text to Speech](#).

Next steps

- [Troubleshooting](#)
- [Release notes](#)

Quickstart: Recognize speech, intents, and entities with Language Understanding (LUIS)

1/3/2020 • 25 minutes to read • [Edit Online](#)

In this quickstart, you'll use the [Speech SDK](#) and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `IntentRecognizer` object using the `SpeechConfig` object from above.
- Using the `IntentRecognizer` object, start the recognition process for a single utterance.
- Inspect the `IntentRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started:

- If this is your first C# project, use this guide to [create an empty sample project](#).
- [Install the Speech SDK for your development environment](#).

Create a LUIS app for intent recognition

To complete the intent recognition quickstart, you'll need to create a LUIS account and a project using the LUIS preview portal. This quickstart only requires a LUIS subscription. A Speech service subscription isn't required.

The first thing you'll need to do is create a LUIS account and app using the LUIS preview portal. The LUIS app that you create will use a prebuilt domain for home automation, which provides intents, entities, and example utterances. When you're finished, you'll have a LUIS endpoint running in the cloud that you can call using the Speech SDK.

Follow these instructions to create your LUIS app:

- [Quickstart: Build prebuilt domain app](#)

When you're done, you'll need three things:

- Your LUIS key
- Your LUIS region
- Your LUIS app ID

Here's where you can find this information in the [LUIS preview portal](#):

1. From the LUIS preview portal, select **Manage**, then select **Azure Resources**. On this page, you'll find your LUIS key and location (sometimes referred to as *region*).

intent-recognition (V0.1) ▾

DASHBOARD BUILD MANAGE Train Publish Test

Application Settings

Publish Settings

Azure Resources

Versions

Location: westus

Primary Key: xxxxxxxxxxxxxxxxxxxxxxxxx

Secondary Key: xxxxxxxxxxxxxxxxxxxxxxxxx

Endpoint URL: https://contoso.cognitiveservices.azure.com/

Pricing Tier: S0 (Standard)

- After you've got your key and location, you'll need the app ID. Select **Application Settings** -- your app ID is available on this page.

intent-recognition (V0.1) ▾

DASHBOARD BUILD MANAGE Train Publish Test

Application Settings

Publish Settings

Azure Resources

Versions

Name * intent-recognition

Description Intent recognition with the Speech SDK

App ID xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

Culture en-us

Open your project in Visual Studio

Next, open your project in Visual Studio.

- Launch Visual Studio 2019.
- Load your project and open `Program.cs`.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project. Make note that you've created an async method called `RecognizeIntentAsync()`.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Intent;

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeIntentAsync()
        {

        }

        static void Main()
        {
            RecognizeIntentAsync().Wait();
            Console.WriteLine("Please press <Return> to continue.");
            Console.ReadLine();
        }
    }
}
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and

location for your LUIS prediction resource.

IMPORTANT

Your starter key and authoring keys will not work. You must use your prediction key and location that you created earlier. For more information, see [Create a LUIS app for intent recognition](#).

Insert this code in the `RecognizeIntentAsync()` method. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location.

TIP

If you need help finding these values, see [Create a LUIS app for intent recognition](#).

```
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
    "YourLanguageUnderstandingServiceRegion");
```

This sample uses the `FromSubscription()` method to build the `SpeechConfig`. For a full list of available methods, see [SpeechConfig Class](#).

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. This object is created inside of a using statement to ensure the proper release of unmanaged resources. Insert this code in the `RecognizeIntentAsync()` method, right below your Speech configuration.

```
// Creates an intent recognizer using microphone as audio input.
using (var recognizer = new IntentRecognizer(config))
{
}
```

Add a LanguageUnderstandingModel and intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer, and add the intents that you want recognized. We're going to use intents from the prebuilt domain for home automation. Insert this code in the using statement from the previous section. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

TIP

If you need help finding this value, see [Create a LUIS app for intent recognition](#).

```
// Creates a Language Understanding model using the app id, and adds specific intents from your model
var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");
```

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech.

Inside the using statement, add this code below your model:

```
var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);
```

Display recognition results (or errors)

When the recognition result is returned by the Speech service, you'll want to do something with it. We're going to keep it simple and print the results to console.

Inside the using statement, below `RecognizeOnceAsync()`, add this code:

```
// Checks result.
if (result.Reason == ResultReason.RecognizedIntent)
{
    Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    Console.WriteLine($"    Intent Id: {result.IntentId}.");
    Console.WriteLine($"    Language Understanding JSON:");
{result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)}.");
}
else if (result.Reason == ResultReason.RecognizedSpeech)
{
    Console.WriteLine($"RECOGNIZED: Text={result.Text}");
    Console.WriteLine($"    Intent not recognized.");
}
else if (result.Reason == ResultReason.NoMatch)
{
    Console.WriteLine($"NOMATCH: Speech could not be recognized.");
}
else if (result.Reason == ResultReason.Canceled)
{
    var cancellation = CancellationDetails.FromResult(result);
    Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

    if (cancellation.Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }
}
```

Check your code

At this point, your code should look like this:

NOTE

We've added some comments to this version.

```
// <code>
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Intent;

namespace helloworld
{
    class Program
    {
        public static async Task RecognizeIntentAsync()
        {
            // Creates an instance of a speech config with specified subscription key
            // and service region. Note that in contrast to other services supported by
            // the Cognitive Services Speech SDK, the Language Understanding service
            // requires a specific subscription key from https://www.luis.ai/.
            // The Language Understanding service calls the required key 'endpoint key'.
            // Once you've obtained it, replace with below with your own Language Understanding subscription
key
            // and service region (e.g., "westus").
            // The default language is "en-us".
            var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

            // Creates an intent recognizer using microphone as audio input.
            using (var recognizer = new IntentRecognizer(config))
            {
                // Creates a Language Understanding model using the app id, and adds specific intents from
your model
                var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
                recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
                recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
                recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

                // Starts recognizing.
                Console.WriteLine("Say something...");

                // Starts intent recognition, and returns after a single utterance is recognized. The end of
a
                // single utterance is determined by listening for silence at the end or until a maximum of
15
                // seconds of audio is processed. The task returns the recognition text as result.
                // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for
single
                // shot recognition like command or query.
                // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync()
instead.

                var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);

                // Checks result.
                if (result.Reason == ResultReason.RecognizedIntent)
                {
                    Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                    Console.WriteLine($"    Intent Id: {result.IntentId}.");
                    Console.WriteLine($"    Language Understanding JSON:
{result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)}.");
                }
                else if (result.Reason == ResultReason.RecognizedSpeech)
                {
                    Console.WriteLine($"RECOGNIZED: Text={result.Text}");
                    Console.WriteLine($"    Intent not recognized.");
                }
            }
        }
    }
}
```

```

        }
        else if (result.Reason == ResultReason.NoMatch)
        {
            Console.WriteLine($"NOMATCH: Speech could not be recognized.");
        }
        else if (result.Reason == ResultReason.Canceled)
        {
            var cancellation = CancellationDetails.FromResult(result);
            Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

            if (cancellation.Reason == CancellationReason.Error)
            {
                Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
                Console.WriteLine($"CANCELED: Did you update the subscription info?");
            }
        }
    }

    static void Main()
    {
        RecognizeIntentAsync().Wait();
        Console.WriteLine("Please press <Return> to continue.");
        Console.ReadLine();
    }
}
}

```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

- 1. Compile the code** - From the menu bar of Visual Studio, choose **Build > Build Solution**.
- 2. Start your app** - From the menu bar, choose **Debug > Start Debugging** or press **F5**.
- 3. Start recognition** - It'll prompt you to speak a phrase in English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

Next steps

[Explore C# samples on GitHub](#)

In this quickstart, you'll use the [Speech SDK](#) and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `IntentRecognizer` object using the `SpeechConfig` object from above.
- Using the `IntentRecognizer` object, start the recognition process for a single utterance.
- Inspect the `IntentRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started:

- If this is your first C++ project, use this guide to [create an empty sample project](#).
- [Install the Speech SDK for your development environment](#).

Create a LUIS app for intent recognition

To complete the intent recognition quickstart, you'll need to create a LUIS account and a project using the LUIS preview portal. This quickstart only requires a LUIS subscription. A Speech service subscription isn't required.

The first thing you'll need to do is create a LUIS account and app using the LUIS preview portal. The LUIS app that you create will use a prebuilt domain for home automation, which provides intents, entities, and example utterances. When you're finished, you'll have a LUIS endpoint running in the cloud that you can call using the Speech SDK.

Follow these instructions to create your LUIS app:

- [Quickstart: Build prebuilt domain app](#)

When you're done, you'll need three things:

- Your LUIS key
- Your LUIS region
- Your LUIS app ID

Here's where you can find this information in the [LUIS preview portal](#):

1. From the LUIS preview portal, select **Manage**, then select **Azure Resources**. On this page, you'll find your LUIS key and location (sometimes referred to as *region*).

Setting	Value
Location	westus
Primary Key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Secondary Key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Endpoint URL	https://contoso.cognitiveservices.azure.com/
Pricing Tier	S0 (Standard)

2. After you've got your key and location, you'll need the app ID. Select **Application Settings** -- your app ID is available on this page.

Setting	Value
Name *	intent-recognition
App ID	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Description	Intent recognition with the Speech SDK
Culture	en-us

Open your project in Visual Studio

Next, open your project in Visual Studio.

1. Launch Visual Studio 2019.
2. Load your project and open `helloworld.cpp`.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project. Make note that you've created an async method called `recognizeIntent()`.

```
#include "stdafx.h"
// <code>
#include <iostream>
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Intent;

void recognizeIntent()
{
}

int wmain()
{
    recognizeIntent();
    cout << "Please press a key to continue.\n";
    cin.get();
    return 0;
}
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and location for your LUIS prediction resource.

IMPORTANT

Your starter key and authoring keys will not work. You must use your prediction key and location that you created earlier. For more information, see [Create a LUIS app for intent recognition](#).

Insert this code in the `recognizeIntent()` method. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location.

TIP

If you need help finding these values, see [Create a LUIS app for intent recognition](#).

```
auto config = SpeechConfig::FromSubscription("YourLanguageUnderstandingSubscriptionKey",
    "YourLanguageUnderstandingServiceRegion");
```

This sample uses the `FromSubscription()` method to build the `SpeechConfig`. For a full list of available methods, see [SpeechConfig Class](#).

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. Insert this code in the `recognizeIntent()` method, right below your Speech configuration.

```
auto recognizer = IntentRecognizer::FromConfig(config);
```

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer, and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

TIP

If you need help finding this value, see [Create a LUIS app for intent recognition](#).

```
auto model = LanguageUnderstandingModel::FromAppId("YourLanguageUnderstandingAppId");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");
```

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `RecognizeOnceAsync()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech. For simplicity we'll wait on the future returned to complete.

Insert this code below your model:

```
auto result = recognizer->RecognizeOnceAsync().get();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, you'll want to do something with it. We're going to keep it simple and print the result to console.

Insert this code below `auto result = recognizer->RecognizeOnceAsync().get();`:

```

if (result->Reason == ResultReason::RecognizedIntent)
{
    cout << "RECOGNIZED: Text=" << result->Text << std::endl;
    cout << " Intent Id: " << result->IntentId << std::endl;
    cout << " Intent Service JSON: " << result-
>Properties.GetProperty(PropertyId::LanguageUnderstandingServiceResponse_JsonResult) << std::endl;
}
else if (result->Reason == ResultReason::RecognizedSpeech)
{
    cout << "RECOGNIZED: Text=" << result->Text << " (intent could not be recognized)" << std::endl;
}
else if (result->Reason == ResultReason::NoMatch)
{
    cout << "NOMATCH: Speech could not be recognized." << std::endl;
}
else if (result->Reason == ResultReason::Canceled)
{
    auto cancellation = CancellationDetails::FromResult(result);
    cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

    if (cancellation->Reason == CancellationReason::Error)
    {
        cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
        cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
        cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}
}

```

Check your code

At this point, your code should look like this:

NOTE

We've added some comments to this version.

```

#include "stdafx.h"
// <code>
#include <iostream>
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Intent;

void recognizeIntent()
{
    // Creates an instance of a speech config with specified subscription key
    // and service region. Note that in contrast to other services supported by
    // the Cognitive Services Speech SDK, the Language Understanding service
    // requires a specific subscription key from https://www.luis.ai/.
    // The Language Understanding service calls the required key 'endpoint key'.
    // Once you've obtained it, replace with below with your own Language Understanding subscription key
    // and service region (e.g., "westus").
    // The default recognition language is "en-us".
    auto config = SpeechConfig::FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

    // Creates an intent recognizer using microphone as audio input.
    auto recognizer = IntentRecognizer::FromConfig(config);

    // Creates a Language Understanding model using the app id, and adds specific intents from your model
    auto model = LanguageUnderstandingModel::FromAppId("YourLanguageUnderstandingAppId");
}

```

```

recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
recognizer->AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

cout << "Say something...\n";

// Starts intent recognition, and returns after a single utterance is recognized. The end of a
// single utterance is determined by listening for silence at the end or until a maximum of 15
// seconds of audio is processed. The task returns the recognition text as result.
// Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
// shot recognition like command or query.
// For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
auto result = recognizer->RecognizeOnceAsync().get();

// Checks result.
if (result->Reason == ResultReason::RecognizedIntent)
{
    cout << "RECOGNIZED: Text=" << result->Text << std::endl;
    cout << " Intent Id: " << result->IntentId << std::endl;
    cout << " Intent Service JSON: " << result-
>Properties.GetProperty(PropertyId::LanguageUnderstandingServiceResponse_JsonResult) << std::endl;
}
else if (result->Reason == ResultReason::RecognizedSpeech)
{
    cout << "RECOGNIZED: Text=" << result->Text << " (intent could not be recognized)" << std::endl;
}
else if (result->Reason == ResultReason::NoMatch)
{
    cout << "NOMATCH: Speech could not be recognized." << std::endl;
}
else if (result->Reason == ResultReason::Canceled)
{
    auto cancellation = CancellationDetails::FromResult(result);
    cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

    if (cancellation->Reason == CancellationReason::Error)
    {
        cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
        cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
        cout << "CANCELED: Did you update the subscription info?" << std::endl;
    }
}
}

int wmain()
{
    recognizeIntent();
    cout << "Please press a key to continue.\n";
    cin.get();
    return 0;
}

```

Build and run your app

Now you're ready to build your app and test our speech recognition using the Speech service.

- Compile the code** - From the menu bar of Visual Studio, choose **Build > Build Solution**.
- Start your app** - From the menu bar, choose **Debug > Start Debugging** or press **F5**.
- Start recognition** - It'll prompt you to speak a phrase in English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

Next steps

[Explore C++ samples on GitHub](#)

In this quickstart, you'll use the [Speech SDK](#) and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `IntentRecognizer` object using the `SpeechConfig` object from above.
- Using the `IntentRecognizer` object, start the recognition process for a single utterance.
- Inspect the `IntentRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started:

- If this is your first Java (JRE) project, use this guide to [create an empty sample project](#).
- [Install the Speech SDK for your development environment](#).

Create a LUIS app for intent recognition

To complete the intent recognition quickstart, you'll need to create a LUIS account and a project using the LUIS preview portal. This quickstart only requires a LUIS subscription. A Speech service subscription isn't required.

The first thing you'll need to do is create a LUIS account and app using the LUIS preview portal. The LUIS app that you create will use a prebuilt domain for home automation, which provides intents, entities, and example utterances. When you're finished, you'll have a LUIS endpoint running in the cloud that you can call using the Speech SDK.

Follow these instructions to create your LUIS app:

- [Quickstart: Build prebuilt domain app](#)

When you're done, you'll need three things:

- Your LUIS key
- Your LUIS region
- Your LUIS app ID

Here's where you can find this information in the [LUIS preview portal](#):

1. From the LUIS preview portal, select **Manage**, then select **Azure Resources**. On this page, you'll find your LUIS key and location (sometimes referred to as *region*).

The screenshot shows the LUIS preview portal interface. At the top, there is a navigation bar with tabs: DASHBOARD, BUILD, MANAGE (which is highlighted in red), Train, Publish, and Test. Below the navigation bar, there is a sidebar with links: Application Settings, Publish Settings, Azure Resources (which is highlighted in red), and Versions. The main content area displays application settings for the 'intent-recognition (V0.1)' project. It includes fields for Location (set to 'westus'), Primary Key (a long string of characters), Secondary Key (another long string of characters), Endpoint URL ('https://contoso.cognitiveservices.azure.com/'), and Pricing Tier ('S0 (Standard)').

2. After you've got your key and location, you'll need the app ID. Select **Application Settings** -- your app ID is available on this page.

The screenshot shows the Azure portal interface for managing a project. The top navigation bar includes 'DASHBOARD', 'BUILD', 'MANAGE' (which is highlighted with a red box), 'Train' (with a red dot), 'Publish', and 'Test'. On the left, a sidebar lists 'Application Settings', 'Publish Settings', 'Azure Resources', and 'Versions'. The main content area is titled 'Application Settings' and contains fields for 'Name *' (set to 'intent-recognition'), 'App ID' (set to 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX'), 'Description' (set to 'Intent recognition with the Speech SDK'), and 'Culture' (set to 'en-us').

Open your project

1. Open your preferred IDE.
2. Load your project and open `Main.java`.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project.

```
package speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.intent.*;

/**
 * Quickstart: recognize speech using the Speech SDK for Java.
 */
public class Main {

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {
        try {
        } catch (Exception ex) {
            System.out.println("Unexpected exception: " + ex.getMessage());

            assert(false);
            System.exit(1);
        }
    }
}
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and location for your LUIS prediction resource.

Insert this code in the try / catch block in `main()`. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location.

TIP

If you need help finding these values, see [Create a LUIS app for intent recognition](#).

```
SpeechConfig config = SpeechConfig.fromSubscription("YourLanguageUnderstandingSubscriptionKey",  
"YourLanguageUnderstandingServiceRegion");
```

This sample uses the `FromSubscription()` method to build the `SpeechConfig`. For a full list of available methods, see [SpeechConfig Class](#).

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. Insert this code right below your Speech configuration.

```
IntentRecognizer recognizer = new IntentRecognizer(config);
```

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer, and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

TIP

If you need help finding this value, see [Create a LUIS app for intent recognition](#).

```
LanguageUnderstandingModel model = LanguageUnderstandingModel.fromAppId("YourLanguageUnderstandingAppId");  
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName1", "id1");  
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName2", "id2");  
recognizer.addIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");
```

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `recognizeOnceAsync()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech.

Insert this code below your model:

```
IntentRecognitionResult result = recognizer.recognizeOnceAsync().get();
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, you'll want to do something with it. We're going to keep it simple and print the result to console.

Insert this code below your call to `recognizeOnceAsync()`:

```

if (result.getReason() == ResultReason.RecognizedIntent) {
    System.out.println("RECOGNIZED: Text=" + result.getText());
    System.out.println("    Intent Id: " + result.getId());
    System.out.println("    Intent Service JSON: " +
result.getProperties().getProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult));
}
else if (result.getReason() == ResultReason.RecognizedSpeech) {
    System.out.println("RECOGNIZED: Text=" + result.getText());
    System.out.println("    Intent not recognized.");
}
else if (result.getReason() == ResultReason.NoMatch) {
    System.out.println("NOMATCH: Speech could not be recognized.");
}
else if (result.getReason() == ResultReason.Canceled) {
    CancellationDetails cancellation = CancellationDetails.fromResult(result);
    System.out.println("CANCELED: Reason=" + cancellation.getReason());

    if (cancellation.getReason() == CancellationReason.Error) {
        System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
        System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
        System.out.println("CANCELED: Did you update the subscription info?");
    }
}

```

Release Resources

It's important to release the speech resources when you're done using them. Insert this code at the end of the try / catch block:

```

result.close();
recognizer.close();

```

Check your code

At this point, your code should look like this:

NOTE

We've added some comments to this version.

```

package speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.intent.*;

/**
 * Quickstart: recognize speech using the Speech SDK for Java.
 */
public class Main {

    /**
     * @param args Arguments are ignored in this sample.
     */
    public static void main(String[] args) {
        try {
            // <IntentRecognitionWithMicrophone>
            // Creates an instance of a speech config with specified
            // subscription key (called 'endpoint key' by the Language Understanding service)
            // and service region. Replace with your own subscription (endpoint) key
            // and service region (e.g., "westus2").

```

```

        // The default language is "en-us".
        SpeechConfig config = SpeechConfig.fromSubscription("YourLanguageUnderstandingSubscriptionKey",
        "YourLanguageUnderstandingServiceRegion");

        // Creates an intent recognizer using microphone as audio input.
        IntentRecognizer recognizer = new IntentRecognizer(config);

        // Creates a language understanding model using the app id, and adds specific intents from your
model
        LanguageUnderstandingModel model =
LanguageUnderstandingModel.fromAppId("YourLanguageUnderstandingAppId");
        recognizer.addIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
        recognizer.addIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
        recognizer.addIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

        System.out.println("Say something...");

        // Starts recognition. It returns when the first utterance has been recognized.
        IntentRecognitionResult result = recognizer.recognizeOnceAsync().get();

        // Checks result.
        if (result.getReason() == ResultReason.RecognizedIntent) {
            System.out.println("RECOGNIZED: Text=" + result.getText());
            System.out.println("    Intent Id: " + result.getIdent());
            System.out.println("    Intent Service JSON: " +
result.getProperties().getProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult));
        }
        else if (result.getReason() == ResultReason.RecognizedSpeech) {
            System.out.println("RECOGNIZED: Text=" + result.getText());
            System.out.println("    Intent not recognized.");
        }
        else if (result.getReason() == ResultReason.NoMatch) {
            System.out.println("NOMATCH: Speech could not be recognized.");
        }
        else if (result.getReason() == ResultReason.Canceled) {
            CancellationDetails cancellation = CancellationDetails.fromResult(result);
            System.out.println("CANCELED: Reason=" + cancellation.getReason());

            if (cancellation.getReason() == CancellationReason.Error) {
                System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
                System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
                System.out.println("CANCELED: Did you update the subscription info?");
            }
        }

        result.close();
        recognizer.close();
    } catch (Exception ex) {
        System.out.println("Unexpected exception: " + ex.getMessage());

        assert(false);
        System.exit(1);
    }
}
}

```

Build and run your app

Press F11, or select **Run > Debug**. The next 15 seconds of speech input from your microphone will be recognized and logged in the console window.

Next steps

[Explore Java samples on GitHub](#)

In this quickstart, you'll use the [Speech SDK](#) and the Language Understanding (LUIS) service to recognize intents from audio data captured from a microphone. Specifically, you'll use the Speech SDK to capture speech, and a prebuilt domain from LUIS to identify intents for home automation, like turning on and off a light.

After satisfying a few prerequisites, recognizing speech and identifying intents from a microphone only takes a few steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Create an `IntentRecognizer` object using the `SpeechConfig` object from above.
- Using the `IntentRecognizer` object, start the recognition process for a single utterance.
- Inspect the `IntentRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started:

- If this is your first Python project, use this guide to [create an empty sample project](#).
- [Install the Speech SDK for your development environment](#).

Create a LUIS app for intent recognition

To complete the intent recognition quickstart, you'll need to create a LUIS account and a project using the LUIS preview portal. This quickstart only requires a LUIS subscription. A Speech service subscription isn't required.

The first thing you'll need to do is create a LUIS account and app using the LUIS preview portal. The LUIS app that you create will use a prebuilt domain for home automation, which provides intents, entities, and example utterances. When you're finished, you'll have a LUIS endpoint running in the cloud that you can call using the Speech SDK.

Follow these instructions to create your LUIS app:

- [Quickstart: Build prebuilt domain app](#)

When you're done, you'll need three things:

- Your LUIS key
- Your LUIS region
- Your LUIS app ID

Here's where you can find this information in the [LUIS preview portal](#):

1. From the LUIS preview portal, select **Manage**, then select **Azure Resources**. On this page, you'll find your LUIS key and location (sometimes referred to as *region*).

The screenshot shows the LUIS preview portal interface. At the top, there is a navigation bar with tabs: DASHBOARD, BUILD, MANAGE (which is highlighted with a red box), Train, Publish, and Test. Below the navigation bar, there are several sections: Application Settings, Publish Settings, Azure Resources (which is also highlighted with a red box), and Versions. The Azure Resources section contains fields for Location (set to westus), Primary Key (a long string of characters), Secondary Key (another long string of characters), Endpoint URL (https://contoso.cognitiveservices.azure.com/), and Pricing Tier (S0 (Standard)).

2. After you've got your key and location, you'll need the app ID. Select **Application Settings** -- your app ID is available on this page.

The screenshot shows the Azure portal interface for managing a project. The top navigation bar includes 'DASHBOARD', 'BUILD', 'MANAGE' (which is highlighted with a red box), 'Train' (with a red dot), 'Publish', and 'Test'. On the left, a sidebar lists 'Application Settings', 'Publish Settings', 'Azure Resources', and 'Versions'. The main content area is titled 'Application Settings' and contains fields for 'Name *' (set to 'intent-recognition'), 'App ID' (containing a long GUID with a copy icon), 'Description' ('Intent recognition with the Speech SDK'), and 'Culture' ('en-us').

Open your project

1. Open your preferred IDE.
2. Create a new project and create file called `quickstart.py`, then open it.

Start with some boilerplate code

Let's add some code that works as a skeleton for our project.

```
import azure.cognitiveservices.speech as speechsdk

print("Say something...")
```

Create a Speech configuration

Before you can initialize an `IntentRecognizer` object, you need to create a configuration that uses the key and location for your LUIS prediction resource.

Insert this code in `quickstart.py`. Make sure you update these values:

- Replace `"YourLanguageUnderstandingSubscriptionKey"` with your LUIS prediction key.
- Replace `"YourLanguageUnderstandingServiceRegion"` with your LUIS location.

TIP

If you need help finding these values, see [Create a LUIS app for intent recognition](#).

```
intent_config = speechsdk.SpeechConfig(subscription="YourLanguageUnderstandingSubscriptionKey",
region="YourLanguageUnderstandingServiceRegion")
```

This sample constructs the `SpeechConfig` object using LUIS key and region. For a full list of available methods, see [SpeechConfig Class](#).

The Speech SDK will default to recognizing using en-us for the language, see [Specify source language for speech to text](#) for information on choosing the source language.

Initialize an IntentRecognizer

Now, let's create an `IntentRecognizer`. Insert this code right below your Speech configuration.

```
intent_recognizer = speechsdk.intent.IntentRecognizer(speech_config=intent_config)
```

Add a LanguageUnderstandingModel and Intents

You need to associate a `LanguageUnderstandingModel` with the intent recognizer and add the intents you want recognized. We're going to use intents from the prebuilt domain for home automation.

Insert this code below your `IntentRecognizer`. Make sure that you replace `"YourLanguageUnderstandingAppId"` with your LUIS app ID.

TIP

If you need help finding this value, see [Create a LUIS app for intent recognition](#).

```
model = speechsdk.intent.LanguageUnderstandingModel(app_id="YourLanguageUnderstandingAppId")
intents = [
    (model, "HomeAutomation.TurnOn"),
    (model, "HomeAutomation.TurnOff"),
    ("This is a test.", "test"),
    ("Switch to channel 34.", "34"),
    ("what's the weather like", "weather"),
]
intent_recognizer.add_intents(intents)
```

Recognize an intent

From the `IntentRecognizer` object, you're going to call the `recognize_once()` method. This method lets the Speech service know that you're sending a single phrase for recognition, and that once the phrase is identified to stop recognizing speech.

Insert this code below your model:

```
intent_result = intent_recognizer.recognize_once()
```

Display the recognition results (or errors)

When the recognition result is returned by the Speech service, you'll want to do something with it. We're going to keep it simple and print the result to console.

Below your call to `recognize_once()`, add this code:

```
if intent_result.reason == speechsdk.ResultReason.RecognizedIntent:
    print("Recognized: \"{}\" with intent id `{}`.".format(intent_result.text, intent_result.intent_id))
elif intent_result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("Recognized: {}".format(intent_result.text))
elif intent_result.reason == speechsdk.ResultReason.NoMatch:
    print("No speech could be recognized: {}".format(intent_result.no_match_details))
elif intent_result.reason == speechsdk.ResultReason.Canceled:
    print("Intent recognition canceled: {}".format(intent_result.cancellation_details.reason))
    if intent_result.cancellation_details.reason == speechsdk.CancellationReason.Error:
        print("Error details: {}".format(intent_result.cancellation_details.error_details))
```

Check your code

At this point, your code should look like this:

NOTE

We've added some comments to this version.

```
import azure.cognitiveservices.speech as speechsdk

print("Say something...")

"""performs one-shot intent recognition from input from the default microphone"""
# <IntentRecognitionOnceWithMic>
# Set up the config for the intent recognizer (remember that this uses the Language Understanding key, not
# the Speech Services key)!
intent_config = speechsdk.SpeechConfig(subscription="YourLanguageUnderstandingSubscriptionKey",
region="YourLanguageUnderstandingServiceRegion")

# Set up the intent recognizer
intent_recognizer = speechsdk.intent.IntentRecognizer(speech_config=intent_config)

# set up the intents that are to be recognized. These can be a mix of simple phrases and
# intents specified through a LanguageUnderstanding Model.
model = speechsdk.intent.LanguageUnderstandingModel(app_id="YourLanguageUnderstandingAppId")
intents = [
    (model, "HomeAutomation.TurnOn"),
    (model, "HomeAutomation.TurnOff"),
    ("This is a test.", "test"),
    ("Switch to channel 34.", "34"),
    ("what's the weather like", "weather"),
]
intent_recognizer.add_intents(intents)

# Starts intent recognition, and returns after a single utterance is recognized. The end of a
# single utterance is determined by listening for silence at the end or until a maximum of 15
# seconds of audio is processed. It returns the recognition text as result.
# Note: Since recognize_once() returns only a single utterance, it is suitable only for single
# shot recognition like command or query.
# For long-running multi-utterance recognition, use start_continuous_recognition() instead.
intent_result = intent_recognizer.recognize_once()

# Check the results
if intent_result.reason == speechsdk.ResultReason.RecognizedIntent:
    print("Recognized: \"{}\" with intent id `{}`.".format(intent_result.text, intent_result.intent_id))
elif intent_result.reason == speechsdk.ResultReason.RecognizedSpeech:
    print("Recognized: {}".format(intent_result.text))
elif intent_result.reason == speechsdk.ResultReason.NoMatch:
    print("No speech could be recognized: {}".format(intent_result.no_match_details))
elif intent_result.reason == speechsdk.ResultReason.Canceled:
    print("Intent recognition canceled: {}".format(intent_result.cancellation_details.reason))
    if intent_result.cancellation_details.reason == speechsdk.CancellationReason.Error:
        print("Error details: {}".format(intent_result.cancellation_details.error_details))
```

Build and run your app

Run the sample from the console or in your IDE:

```
python quickstart.py
```

The next 15 seconds of speech input from your microphone will be recognized and logged in the console window.

Next steps

Explore Python samples on GitHub

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	ADDITIONAL QUICKSTARTS	CODE SAMPLES
C++		Quickstarts , Samples
C#		.NET Framework , .NET Core , UWP , Unity , Xamarin
Java		Android , JRE
Javascript		Browser
Node.js		Windows , Linux , macOS
Objective-C	macOs , iOS	iOS , macOS
Python		Windows , Linux , macOS
Swift	macOs , iOS	iOS , macOS

How to recognize intents from speech using the Speech SDK for C#

12/4/2019 • 12 minutes to read • [Edit Online](#)

The Cognitive Services [Speech SDK](#) integrates with the [Language Understanding service \(LUIS\)](#) to provide **intent recognition**. An intent is something the user wants to do: book a flight, check the weather, or make a call. The user can use whatever terms feel natural. Using machine learning, LUIS maps user requests to the intents you've defined.

NOTE

A LUIS application defines the intents and entities you want to recognize. It's separate from the C# application that uses the Speech service. In this article, "app" means the LUIS app, while "application" means the C# code.

In this guide, you use the Speech SDK to develop a C# console application that derives intents from user utterances through your device's microphone. You'll learn how to:

- Create a Visual Studio project referencing the Speech SDK NuGet package
- Create a speech configuration and get an intent recognizer
- Get the model for your LUIS app and add the intents you need
- Specify the language for speech recognition
- Recognize speech from a file
- Use asynchronous, event-driven continuous recognition

Prerequisites

Be sure you have the following items before you begin this guide:

- A LUIS account. You can get one for free through the [LUIS portal](#).
- [Visual Studio 2019](#) (any edition).

LUIS and speech

LUIS integrates with the Speech service to recognize intents from speech. You don't need a Speech service subscription, just LUIS.

LUIS uses three kinds of keys:

KEY TYPE	PURPOSE
Authoring	Lets you create and modify LUIS apps programmatically
Starter	Lets you test your LUIS application using text only
Endpoint	Authorizes access to a particular LUIS app

For this guide, you need the endpoint key type. This guide uses the example Home Automation LUIS app, which you can create by following the [Use prebuilt Home automation app](#) quickstart. If you've created a LUIS app of your own, you can use it instead.

When you create a LUIS app, LUIS automatically generates a starter key so you can test the app using text queries. This key doesn't enable the Speech service integration and won't work with this guide. Create a LUIS resource in the Azure dashboard and assign it to the LUIS app. You can use the free subscription tier for this guide.

After you create the LUIS resource in the Azure dashboard, log into the [LUIS portal](#), choose your application on the **My Apps** page, then switch to the app's **Manage** page. Finally, select **Keys and Endpoints** in the sidebar.

The screenshot shows the LUIS portal interface. At the top, there is a navigation bar with links for Language Understanding, My apps, Docs, Pricing, Support, and About. Below this is a secondary navigation bar for the 'HomeAutomation (V 0.1)' application, with tabs for DASHBOARD, BUILD, and MANAGE. The MANAGE tab is currently selected, indicated by a blue underline. On the left side, there is a sidebar with links for Application Information, Keys and Endpoints (which is highlighted with a blue background), Publish Settings, Versions, and Collaborators. The main content area is titled 'Keys and Endpoint settings' and contains a sub-section titled 'Authoring Key'. A file icon with a progress bar is shown below the sub-section title.

On the **Keys and Endpoint settings** page:

1. Scroll down to the **Resources and Keys** section and select **Assign resource**.
2. In the **Assign a key to your app** dialog box, make the following changes:
 - Under **Tenant**, choose **Microsoft**.
 - Under **Subscription Name**, choose the Azure subscription that contains the LUIS resource you want to use.
 - Under **Key**, choose the LUIS resource that you want to use with the app.In a moment, the new subscription appears in the table at the bottom of the page.
3. Select the icon next to a key to copy it to the clipboard. (You may use either key.)

The screenshot shows a table titled '+ Assign resource' with two rows of data. The columns are labeled: Resource Name, Region, Time zone, Key 1, and Key 2. The first row contains 'Starter_Key' in the Resource Name column, 'westus' in the Region column, 'GMT -6:00' in the Time zone column, and two download icons in the Key 1 and Key 2 columns, respectively. The second row contains 'luis' in the Resource Name column, 'westus' in the Region column, 'GMT -6:00' in the Time zone column, and two download icons in the Key 1 and Key 2 columns, respectively.

Resource Name	Region	Time zone	Key 1	Key 2
Starter_Key	westus	GMT -6:00		
luis	westus	GMT -6:00		

Create a speech project in Visual Studio

To create a Visual Studio project for Windows development, you need to create the project, set up Visual Studio for .NET desktop development, install the Speech SDK, and choose the target architecture.

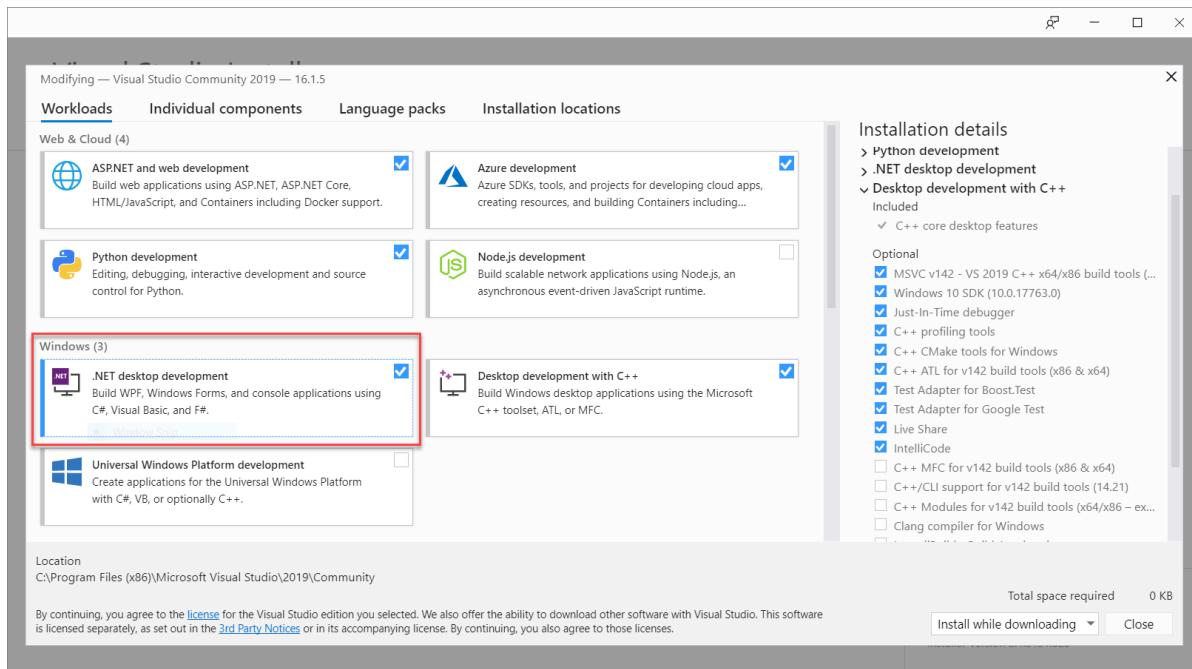
Create the project and add the workload

To start, create the project in Visual Studio, and make sure that Visual Studio is set up for .NET desktop development:

1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.
3. In the **Create a new project** window, choose **Console App (.NET Framework)**, and then select **Next**.

- In the **Configure your new project** window, enter *helloworld* in **Project name**, choose or create the directory path in **Location**, and then select **Create**.
- From the Visual Studio menu bar, select **Tools > Get Tools and Features**, which opens Visual Studio Installer and displays the **Modifying** dialog box.
- Check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, select the check box next to it, and then select **Modify** to start the installation. It may take a few minutes to download and install.

If the check box next to **.NET desktop development** is already selected, select **Close** to exit the dialog box.

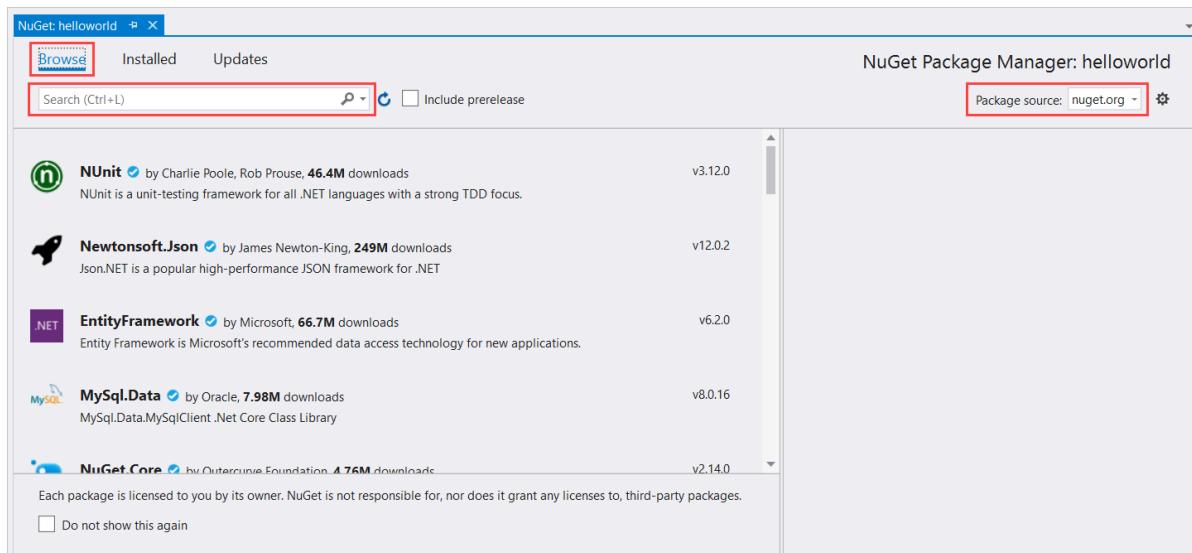


7. Close Visual Studio Installer.

Install the Speech SDK

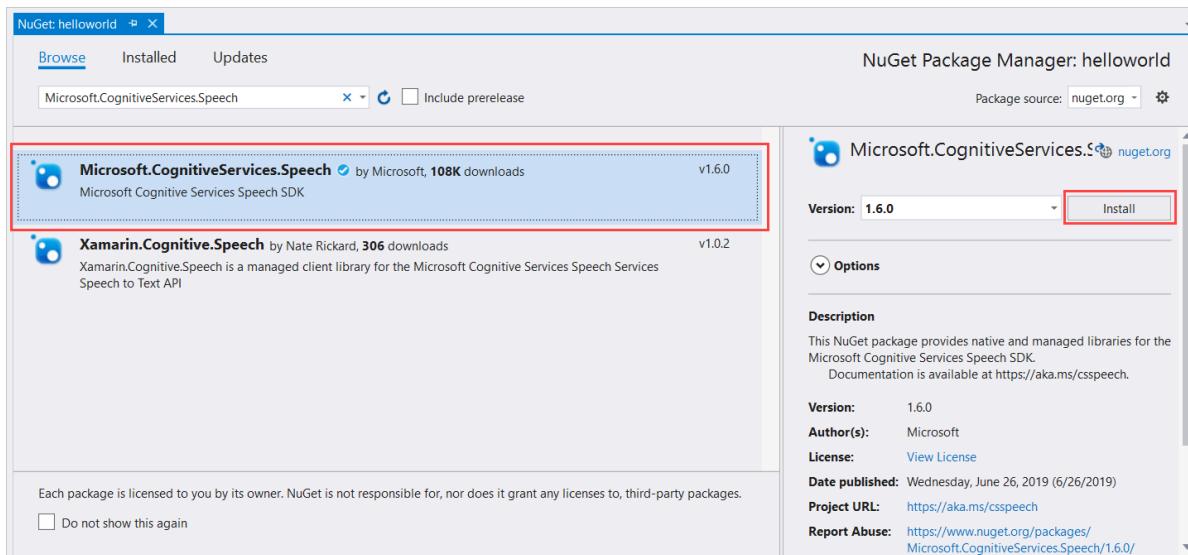
The next step is to install the [Speech SDK NuGet package](#), so you can reference it in the code.

- In the Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show the NuGet Package Manager.



- In the upper-right corner, find the **Package Source** drop-down box, and make sure that **nuget.org** is selected.

3. In the upper-left corner, select **Browse**.
4. In the search box, type *Microsoft.CognitiveServices.Speech* and select **Enter**.
5. From the search results, select the **Microsoft.CognitiveServices.Speech** package, and then select **Install** to install the latest stable version.



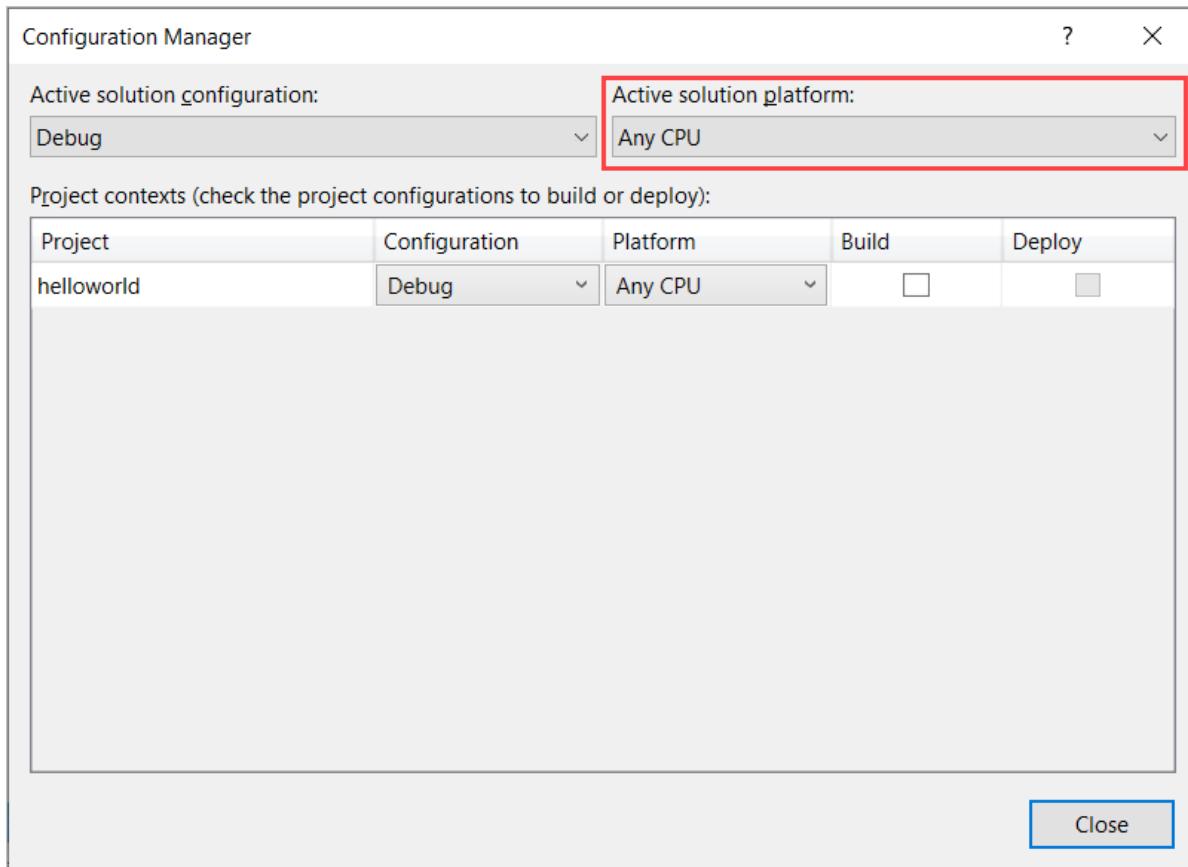
6. Accept all agreements and licenses to start the installation.

After the package is installed, a confirmation appears in the **Package Manager Console** window.

Choose the target architecture

Now, to build and run the console application, create a platform configuration matching your computer's architecture.

1. From the menu bar, select **Build > Configuration Manager**. The **Configuration Manager** dialog box appears.



2. In the **Active solution platform** drop-down box, select **New**. The **New Solution Platform** dialog box appears.
3. In the **Type or select the new platform** drop-down box:
 - If you're running 64-bit Windows, select **x64**.
 - If you're running 32-bit Windows, select **x86**.
4. Select **OK** and then **Close**.

Add the code

Next, you add code to the project.

1. From **Solution Explorer**, open the file **Program.cs**.
2. Replace the block of `using` statements at the beginning of the file with the following declarations:

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Intent;
```

3. Inside the provided `Main()` method, add the following code:

```
RecognizeIntentAsync().Wait();
Console.WriteLine("Please press Enter to continue.");
Console.ReadLine();
```

4. Create an empty asynchronous method `RecognizeIntentAsync()`, as shown here:

```
static async Task RecognizeIntentAsync()
{}
```

5. In the body of this new method, add this code:

```

// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
// The default language is "en-us".
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using microphone as audio input.
using (var recognizer = new IntentRecognizer(config))
{
    // Creates a Language Understanding model using the app id, and adds specific intents from your
    model
    var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

    // Starts recognizing.
    Console.WriteLine("Say something...");

    // Starts intent recognition, and returns after a single utterance is recognized. The end of a
    // single utterance is determined by listening for silence at the end or until a maximum of 15
    // seconds of audio is processed. The task returns the recognition text as result.
    // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
    // shot recognition like command or query.
    // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
    var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);

    // Checks result.
    if (result.Reason == ResultReason.RecognizedIntent)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"    Intent Id: {result.IntentId}.");
        Console.WriteLine($"    Language Understanding JSON:");
        {result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)."};
    }
    else if (result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"    Intent not recognized.");
    }
    else if (result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
    else if (result.Reason == ResultReason.Canceled)
    {
        var cancellation = CancellationDetails.FromResult(result);
        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

        if (cancellation.Reason == CancellationReason.Error)
        {
            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
            Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
            Console.WriteLine($"CANCELED: Did you update the subscription info?");
        }
    }
}

```

- Replace the placeholders in this method with your LUIS subscription key, region, and app ID as follows.

PLACEHOLDER	REPLACE WITH
YourLanguageUnderstandingSubscriptionKey	Your LUIS endpoint key. Again, you must get this item from your Azure dashboard, not a "starter key." You can find it on your app's Keys and Endpoints page (under Manage) in the LUIS portal .
YourLanguageUnderstandingServiceRegion	The short identifier for the region your LUIS subscription is in, such as <code>westus</code> for West US. See Regions .
YourLanguageUnderstandingAppId	The LUIS app ID. You can find it on your app's Settings page in the LUIS portal .

With these changes made, you can build (**Control+Shift+B**) and run (**F5**) the application. When you're prompted, try saying "Turn off the lights" into your PC's microphone. The application displays the result in the console window.

The following sections include a discussion of the code.

Create an intent recognizer

First, you need to create a speech configuration from your LUIS endpoint key and region. You can use speech configurations to create recognizers for the various capabilities of the Speech SDK. The speech configuration has multiple ways to specify the subscription you want to use; here, we use `FromSubscription`, which takes the subscription key and region.

NOTE

Use the key and region of your LUIS subscription, not a Speech service subscription.

Next, create an intent recognizer using `new IntentRecognizer(config)`. Since the configuration already knows which subscription to use, you don't need to specify the subscription key and endpoint again when creating the recognizer.

Import a LUIS model and add intents

Now import the model from the LUIS app using `LanguageUnderstandingModel.FromAppId()` and add the LUIS intents that you wish to recognize via the recognizer's `AddIntent()` method. These two steps improve the accuracy of speech recognition by indicating words that the user is likely to use in their requests. You don't have to add all the app's intents if you don't need to recognize them all in your application.

To add intents, you must provide three arguments: the LUIS model (which has been created and is named `model`), the intent name, and an intent ID. The difference between the ID and the name is as follows.

ADDINTENT() ARGUMENT	PURPOSE
<code>intentName</code>	The name of the intent as defined in the LUIS app. This value must match the LUIS intent name exactly.
<code>intentID</code>	An ID assigned to a recognized intent by the Speech SDK. This value can be whatever you like; it doesn't need to correspond to the intent name as defined in the LUIS app. If multiple intents are handled by the same code, for instance, you could use the same ID for them.

The Home Automation LUIS app has two intents: one for turning on a device, and another for turning off a device. The lines below add these intents to the recognizer; replace the three `AddIntent` lines in the `RecognizeIntentAsync()` method with this code.

```
recognizer.AddIntent(model, "HomeAutomation.TurnOff", "off");
recognizer.AddIntent(model, "HomeAutomation.TurnOn", "on");
```

Instead of adding individual intents, you can also use the `AddAllIntents` method to add all the intents in a model to the recognizer.

Start recognition

With the recognizer created and the intents added, recognition can begin. The Speech SDK supports both single-shot and continuous recognition.

RECOGNITION MODE	METHODS TO CALL	RESULT
Single-shot	<code>RecognizeOnceAsync()</code>	Returns the recognized intent, if any, after one utterance.
Continuous	<code>StartContinuousRecognitionAsync()</code> <code>StopContinuousRecognitionAsync()</code>	Recognizes multiple utterances; emits events (for example, <code>IntermediateResultReceived</code>) when results are available.

The application uses single-shot mode and so calls `RecognizeOnceAsync()` to begin recognition. The result is an `IntentRecognitionResult` object containing information about the intent recognized. You extract the LUIS JSON response by using the following expression:

```
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)
```

The application doesn't parse the JSON result. It only displays the JSON text in the console window.

```
Say something...
RECOGNIZED: Text=Hey turn off the light.
    Intent Id: off.
    Language Understanding JSON: {
        "query": "Hey turn off the light",
        "topScoringIntent": {
            "intent": "HomeAutomation.TurnOff",
            "score": 0.997960269
        },
        "entities": [
            {
                "entity": "light",
                "type": "HomeAutomation.DeviceType",
                "startIndex": 17,
                "endIndex": 21,
                "resolution": {
                    "values": [
                        "light"
                    ]
                }
            }
        ]
    }.
Please press Enter to continue.
```

Specify recognition language

By default, LUIS recognizes intents in US English (`en-us`). By assigning a locale code to the `SpeechRecognitionLanguage` property of the speech configuration, you can recognize intents in other languages. For example, add `config.SpeechRecognitionLanguage = "de-de";` in our application before creating the recognizer to recognize intents in German. For more information, see [Supported Languages](#).

Continuous recognition from a file

The following code illustrates two additional capabilities of intent recognition using the Speech SDK. The first, previously mentioned, is continuous recognition, where the recognizer emits events when results are available. These events can then be processed by event handlers that you provide. With continuous recognition, you call the recognizer's `StartContinuousRecognitionAsync()` method to start recognition instead of `RecognizeOnceAsync()`.

The other capability is reading the audio containing the speech to be processed from a WAV file. Implementation involves creating an audio configuration that can be used when creating the intent recognizer. The file must be single-channel (mono) with a sampling rate of 16 kHz.

To try out these features, delete or comment out the body of the `RecognizeIntentAsync()` method, and add the following code in its place.

```
// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using file as audio input.
// Replace with your own audio file name.
using (var audioInput = AudioConfig.FromWavFileInput("whatstheweatherlike.wav"))
{
    using (var recognizer = new IntentRecognizer(config, audioInput))
    {
        // The TaskCompletionSource to stop recognition.
        var stopRecognition = new TaskCompletionSource<int>();

        // Creates a Language Understanding model using the app id, and adds specific intents from your model
        var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

        // Subscribes to events.
        recognizer.Recognizing += (s, e) => {
            Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
        };

        recognizer.Recognized += (s, e) => {
            if (e.Result.Reason == ResultReason.RecognizedIntent)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"    Intent Id: {e.Result.IntentId}.");
                Console.WriteLine($"    Language Understanding JSON:");
                {e.Result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)}.");
            }
            else if (e.Result.Reason == ResultReason.RecognizedSpeech)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"    Intent not recognized.");
            }
        };
    }
}
```

```

    }

    else if (e.Result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
};

recognizer.Canceled += (s, e) => {
    Console.WriteLine($"CANCELED: Reason={e Reason}");

    if (e Reason == CancellationReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={e.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={e.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");
    }

    stopRecognition.TrySetResult(0);
};

recognizer.SessionStarted += (s, e) => {
    Console.WriteLine("\n    Session started event.");
};

recognizer.SessionStopped += (s, e) => {
    Console.WriteLine("\n    Session stopped event.");
    Console.WriteLine("\nStop recognition.");
    stopRecognition.TrySetResult(0);
};

// Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
Console.WriteLine("Say something...");
await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

// Waits for completion.
// Use Task.WaitAny to keep the task rooted.
Task.WaitAny(new[] { stopRecognition.Task });

// Stops recognition.
await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
}
}
}

```

Revise the code to include your LUIS endpoint key, region, and app ID and to add the Home Automation intents, as before. Change `whatstheweatherlike.wav` to the name of your recorded audio file. Then build, copy the audio file to the build directory, and run the application.

For example, if you say "Turn off the lights", pause, and then say "Turn on the lights" in your recorded audio file, console output similar to the following may appear:

```
Say something...
Session started event.
RECOGNIZING: Text=turn
RECOGNIZING: Text=turn off
RECOGNIZING: Text=turn off the
RECOGNIZING: Text=turn off the light
RECOGNIZING: Text=turn off the lights
RECOGNIZED: Text=Turn off the lights.
    Intent Id: off.
    Language Understanding JSON: {
        "query": "turn off the lights",
        "topScoringIntent": {
            "intent": "HomeAutomation.TurnOff",
            "score": 0.997679055
        },
        "entities": [
            {
                "entity": "lights",
                "type": "HomeAutomation.DeviceType",
                "startIndex": 13,
                "endIndex": 18,
                "resolution": {
                    "values": [
                        "light"
                    ]
                }
            }
        ]
    }.
RECOGNIZING: Text=turn
RECOGNIZING: Text=turn on
RECOGNIZING: Text=turn on the
RECOGNIZING: Text=turn on the light
RECOGNIZING: Text=turn on the lights
RECOGNIZED: Text=Turn on the lights.
    Intent Id: on.
    Language Understanding JSON: {
        "query": "turn on the lights",
        "topScoringIntent": {
            "intent": "HomeAutomation.TurnOn",
            "score": 0.987454832
        },
        "entities": [
            {
                "entity": "lights",
                "type": "HomeAutomation.DeviceType",
                "startIndex": 12,
                "endIndex": 17,
                "resolution": {
                    "values": [
                        "light"
                    ]
                }
            }
        ]
    }.
NOMATCH: Speech could not be recognized.
NOMATCH: Speech could not be recognized.
CANCELED: Reason=EndOfStream

Session stopped event.

Stop recognition.
Please press Enter to continue.
```

Get the samples

For the latest samples, see the [Cognitive Services Speech SDK sample code repository](#) on GitHub.

Look for the code from this article in the **samples/csharp/sharedcontent/console** folder.

Next steps

[Quickstart: Recognize speech from a microphone](#)

Release notes

12/16/2019 • 14 minutes to read • [Edit Online](#)

Speech SDK 1.8.0: 2019-November release

New Features

- Added a `FromHost()` API, to ease use with on-prem containers and sovereign clouds.
- Added Automatic Source Language Detection for Speech Recognition (in Java and C++)
- Added `SourceLanguageConfig` object for Speech Recognition, used to specify expected source languages (in Java and C++)
- Added `KeywordRecognizer` support on Windows (UWP), Android and iOS through the Nuget and Unity packages
- Added Remote Conversation Java API to do Conversation Transcription in asynchronous batches.

Breaking changes

- Conversation Transcriber functionalities moved under namespace
`Microsoft.CognitiveServices.Speech.Transcription`.
- Part of the Conversation Transcriber methods are moved to new `Conversation` class.
- Dropped support for 32-bit (ARMv7 and x86) iOS

Bug fixes

- Fix for crash if local `KeywordRecognizer` is used without a valid speech service subscription key

Samples

- Xamarin sample for `KeywordRecognizer`
- Unity sample for `KeywordRecognizer`
- C++ and Java samples for Automatic Source Language Detection.

Speech SDK 1.7.0: 2019-September release

New Features

- Added beta support for Xamarin on Universal Windows Platform (UWP), Android, and iOS
- Added iOS support for Unity
- Added `Compressed` input support for ALaw, Mulaw, FLAC on Android, iOS and Linux
- Added `SendMessageAsync` in `Connection` class for sending a message to service
- Added `SetMessageProperty` in `Connection` class for setting property of a message
- TTS added bindings for Java (Jre and Android), Python, Swift, and Objective-C
- TTS added playback support for macOS, iOS, and Android.
- Added "word boundary" information for TTS.

Bug fixes

- Fixed IL2CPP build issue on Unity 2019 for Android
- Fixed issue with malformed headers in wav file input being processed incorrectly
- Fixed issue with UUIDs not being unique in some connection properties

- Fixed a few warnings about nullability specifiers in the Swift bindings (might require small code changes)
- Fixed a bug that caused websocket connections to be closed ungracefully under network load
- Fixed an issue on Android that sometimes results in duplicate impression IDs used by `DialogServiceConnector`
- Improvements to the stability of connections across multi-turn interactions and the reporting of failures (via `Canceled` events) when they occur with `DialogServiceConnector`
- `DialogServiceConnector` session starts will now properly provide events, including when calling `ListenOnceAsync()` during an active `StartKeywordRecognitionAsync()`
- Addressed a crash associated with `DialogServiceConnector` activities being received

Samples

- Quickstart for Xamarin
- Updated CPP Quickstart with Linux ARM64 information
- Updated Unity quickstart with iOS information

Speech SDK 1.6.0: 2019-June release

Samples

- Quickstart samples for Text To Speech on UWP and Unity
- Quickstart sample for Swift on iOS
- Unity samples for Speech & Intent Recognition and Translation
- Updated quickstart samples for `DialogServiceConnector`

Improvements / Changes

- Dialog namespace:
 - `SpeechBotConnector` has been renamed to `DialogServiceConnector`
 - `BotConfig` has been renamed to `DialogServiceConfig`
 - `BotConfig::FromChannelSecret()` has been remapped to `DialogServiceConfig::FromBotSecret()`
 - All existing Direct Line Speech clients continue to be supported after the rename
- Update TTS REST adapter to support proxy, persistent connection
- Improve error message when an invalid region is passed
- Swift/Objective-C:
 - Improved error reporting: Methods that can result in an error are now present in two versions: One that exposes an `NSError` object for error handling, and one that raises an exception. The former are exposed to Swift. This change requires adaptations to existing Swift code.
 - Improved event handling

Bug fixes

- Fix for TTS: where `SpeakTextAsync` future returned without waiting until audio has completed rendering
- Fix for marshaling strings in C# to enable full language support
- Fix for .NET core app problem to load core library with net461 target framework in samples
- Fix for occasional issues to deploy native libraries to the output folder in samples
- Fix for web socket closing reliably
- Fix for possible crash while opening a connection under very heavy load on Linux
- Fix for missing metadata in the framework bundle for macOS
- Fix for problems with `pip install --user` on Windows

Speech SDK 1.5.1

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

Speech SDK 1.5.0: 2019-May release

New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see [here](#).
- Conversation transcription functionality is available through the SDK. See [here](#).
- Add support for voice assistants using the Direct Line Speech channel.

Samples

- Added samples for new features or new services supported by the SDK.

Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `outputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

Speech SDK 1.4.0: 2019-April release

New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the [text-to-speech overview](#).
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details [here](#)).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C

support for macOS is currently in beta.

- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.
- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details [here](#)).
- JavaScript: Improve audio processing performance.

Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

Speech SDK 1.3.0: 2019-February release

New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing [audio input device selection](#). This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the [GitHub sample repository](#). This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our [Unity quickstart](#).
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

Samples

The following new content is available in our [sample repository](#):

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `connection` object in iOS.
- Additional Java samples for translation with audio output.

- New sample for use of the [Batch Transcription REST API](#).

Improvements / Changes

- Python
 - Improved parameter verification and error messages in `SpeechConfig`.
 - Add support for the `Connection` object.
 - Support for 32-bit Python (x86) on Windows.
 - The Speech SDK for Python is out of beta.
- iOS
 - The SDK is now built against the iOS SDK version 12.1.
 - The SDK now supports iOS versions 9.2 and later.
 - Improve reference documentation and fix several property names.
- JavaScript
 - Add support for the `Connection` object.
 - Add type definition files for bundled JavaScript
 - Initial support and implementation for phrase hints.
 - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at `turn.end`, not at `speech.end`.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

Speech SDK 1.2.0: 2018-December release

New Features

- Python
 - The Beta version of Python support (3.5 and above) is available with this release. For more information, see [here](#)](quickstart-python.md).
- JavaScript

- The Speech SDK for JavaScript has been open-sourced. The source code is available on [GitHub](#).
- We now support Node.js, more info can be found [here](#).
- The length restriction for audio sessions has been removed, reconnection will happen automatically under the cover.
- `Connection` object
 - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
 - Enabled ProGuard support during APK generation.

Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; `RecognizedIntent` was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the [sample repository](#).

Speech SDK 1.1.0

New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

Improvements

- Added additional verification in the recognizer configuration, and added additional error message.

- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in `AddIntent` in JavaScript, and added missing `AddIntent` JavaScript signature.

Samples

- Added C++ and C# samples for pull and push stream usage in the [sample repository](#).

Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our [sample repository](#), a new sample for JavaScript was added.

Cognitive Services Speech SDK 1.0.0: 2018-September release

New features

- Support for Objective-C on iOS. Check out our [Objective-C quickstart for iOS](#).
- Support for JavaScript in browser. Check out our [JavaScript quickstart](#).

Breaking changes

- With this release, a number of breaking changes are introduced. Check [this page](#) for details.

Cognitive Services Speech SDK 0.6.0: 2018-August release

New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the [UWP quickstart](#).
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the [Java Runtime](#)

[Environment quickstart](#).

Functional change

- Expose additional error detail information on connection errors.

Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.
- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

Cognitive Services Speech SDK 0.5.0: 2018-July release

New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the [Android quickstart](#).
- Support .NET Standard 2.0 on Windows. Check out the [.NET Core quickstart](#).
- Experimental: Support UWP on Windows (version 1709 or later).
 - Check out the [UWP quickstart](#).
 - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support `AuthorizationToken` for creating factory instances.

Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- `SpeechOutputFormat` in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
 - In Java, the `read` method now returns `long` instead of `int`.
 - In C#, the `Read` method now returns `uint` instead of `int`.
 - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a [regions](#) page to describe the supported regions.

Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

Cognitive Services Speech SDK 0.4.0: 2018-June release

Functional changes

- `AudioInputStream`

A recognizer now can consume a stream as the audio source. For more information, see the related [how-to guide](#).

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- Fired a `NoMatch` result when the response from service is timed out.
- The media foundation libraries on Windows are delay loaded. This library is required for microphone input only.
- The upload speed for audio data is limited to about twice the original audio speed.
- On Windows, C# .NET assemblies now are strong named.
- Documentation fix: `Region` is required information to create a recognizer.

More samples have been added and are constantly being updated. For the latest set of samples, see the [Speech SDK samples GitHub repository](#).

Cognitive Services Speech SDK 0.2.12733: 2018-May release

This release is the first public preview release of the Cognitive Services Speech SDK.

What is speech translation?

12/4/2019 • 2 minutes to read • [Edit Online](#)

Speech translation from the Speech service enables real-time, multi-language speech-to-speech and speech-to-text translation of audio streams. With the Speech SDK, your applications, tools, and devices have access to source transcriptions and translation outputs for provided audio. Interim transcription and translation results are returned as speech is detected, and final results can be converted into synthesized speech.

Microsoft's translation engine is powered by two different approaches: statistical machine translation (SMT) and neural machine translation (NMT). SMT uses advanced statistical analysis to estimate the best possible translations given the context of a few words. With NMT, neural networks are used to provide more accurate, natural-sounding translations by using the full context of sentences to translate words.

Today, Microsoft uses NMT for translation to most popular languages. All [languages available for speech-to-speech translation](#) are powered by NMT. Speech-to-text translation may use SMT or NMT depending on the language pair. When the target language is supported by NMT, the full translation is NMT-powered. When the target language isn't supported by NMT, the translation is a hybrid of NMT and SMT, using English as a "pivot" between the two languages.

Core features

Here are the features available via the Speech SDK and REST APIs:

USE CASE	SDK	REST
Speech-to-text translation with recognition results.	Yes	No
Speech-to-speech translation.	Yes	No
Interim recognition and translation results.	Yes	No

Get started with speech translation

We offer quickstarts designed to have you running code in less than 10 minutes. This table includes a list of speech translation quickstarts organized by language.

QUICKSTART	PLATFORM	API REFERENCE
C#, .NET Core	Windows	Browse
C#, .NET Framework	Windows	Browse
C#, UWP	Windows	Browse
C++	Windows	Browse
Java	Windows, Linux, macOS	Browse

Sample code

Sample code for the Speech SDK is available on GitHub. These samples cover common scenarios like reading audio from a file or stream, continuous and single-shot recognition/translation, and working with custom models.

- [Speech-to-text and translation samples \(SDK\)](#)

Migration guides

If your applications, tools, or products are using the [Translator Speech API](#), we've created guides to help you migrate to the Speech service.

- [Migrate from the Translator Speech API to the Speech service](#)

Reference docs

- [Speech SDK](#)
- [Speech Devices SDK](#)
- [REST API: Speech-to-text](#)
- [REST API: Text-to-speech](#)
- [REST API: Batch transcription and customization](#)

Next steps

- [Get a Speech service subscription key for free](#)
- [Get the Speech SDK](#)

Quickstart: Translate speech-to-text

12/4/2019 • 13 minutes to read • [Edit Online](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to text in another language. After satisfying a few prerequisites, translating speech-to-text only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Update the `SpeechConfig` object to specify the source and target languages.
- Create a `TranslationRecognizer` object using the `SpeechConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [.NET](#)
- [.NET Core](#)

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open **Program.cs**, and replace all the code in it with the following.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Translation;

namespace helloworld
{
    class Program
    {
        public static async Task TranslateSpeechToText()
        {
            // Creates an instance of a speech translation config with specified subscription key and
            // service region.
            // Replace with your own subscription key and service region (e.g., "westus").
            var config = SpeechTranslationConfig.FromSubscription("YourSubscriptionKey",
            "YourServiceRegion");

            // Sets source and target languages.
            // Replace with the languages of your choice, from list found here:
            https://aka.ms/speech/sttt-languages
            string fromLanguage = "en-US";
            string toLanguage = "de";
            config.SpeechRecognitionLanguage = fromLanguage;
            config.AddTargetLanguage(toLanguage);
        }
    }
}
```

```

    config.AuditedSpeechLanguage(toLanguage),

    // Creates a translation recognizer using the default microphone audio input device.
    using (var recognizer = new TranslationRecognizer(config))
    {
        // Starts translation, and returns after a single utterance is recognized. The end of
        // a
        // single utterance is determined by listening for silence at the end or until a
        maximum of 15
        // seconds of audio is processed. The task returns the recognized text as well as the
        translation.
        // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable
        only for single
        // shot recognition like command or query.
        // For long-running multi-utterance recognition, use
        StartContinuousRecognitionAsync() instead.
        Console.WriteLine("Say something...");
        var result = await recognizer.RecognizeOnceAsync();

        // Checks result.
        if (result.Reason == ResultReason.TranslatedSpeech)
        {
            Console.WriteLine($"RECOGNIZED '{fromLanguage}': {result.Text}");
            Console.WriteLine($"TRANSLATED into '{toLanguage}'":
{result.Translations[toLanguage]}");
        }
        else if (result.Reason == ResultReason.RecognizedSpeech)
        {
            Console.WriteLine($"RECOGNIZED '{fromLanguage}': {result.Text} (text could not be
translated)");
        }
        else if (result.Reason == ResultReason.NoMatch)
        {
            Console.WriteLine($"NOMATCH: Speech could not be recognized.");
        }
        else if (result.Reason == ResultReason.Canceled)
        {
            var cancellation = CancellationDetails.FromResult(result);
            Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

            if (cancellation.Reason == CancellationReason.Error)
            {
                Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
                Console.WriteLine($"CANCELED: Did you update the subscription info?");
            }
        }
    }

    static void Main(string[] args)
    {
        TranslateSpeechToText().Wait();
    }
}
}

```

2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
4. From the menu bar, choose **File > Save All**.

Build and run the application

1. From the menu bar, select **Build > Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug > Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the text back to the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
```

Next steps

[Explore C# samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to text in another language. After satisfying a few prerequisites, translating speech-to-text only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Update the `SpeechConfig` object to specify the source and target languages.
- Create a `TranslationRecognizer` object using the `SpeechConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open the source file **helloworld.cpp**.
2. Replace all the code with the following snippet:

```
#include <iostream>
#include <vector>
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Translation;

void TranslateSpeechToText()
{
    // Creates an instance of a speech translation config with specified subscription key and service
    // region.
    // Replace with your own subscription key and service region (e.g., "westus").
    auto config = SpeechTranslationConfig::FromSubscription("YourSubscriptionKey",
        "YourServiceRegion");
```

```

YourServiceRegion "",

    // Sets source and target languages.
    // Replace with the languages of your choice, from list found here: https://aka.ms/speech/sttt-
languages
    auto fromLanguage = "en-US";
    auto toLanguage = "de";
    config->SetSpeechRecognitionLanguage(fromLanguage);
    config->AddTargetLanguage(toLanguage);

    // Creates a translation recognizer using the default microphone audio input device.
    auto recognizer = TranslationRecognizer::FromConfig(config);

    // Starts translation, and returns after a single utterance is recognized. The end of a
    // single utterance is determined by listening for silence at the end or until a maximum of 15
    // seconds of audio is processed. The task returns the recognized text as well as the
    translation.
    // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for
single
    // shot recognition like command or query.
    // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
    cout << "Say something...\n";
    auto result = recognizer->RecognizeOnceAsync().get();

    // Checks result.
    if (result->Reason == ResultReason::TranslatedSpeech)
    {
        cout << "RECOGNIZED '" << fromLanguage << "' : " << result->Text << std::endl;
        cout << "TRANSLATED into '" << toLanguage << "' : " << result->Translations.at(toLanguage) <<
std::endl;
    }
    else if (result->Reason == ResultReason::RecognizedSpeech)
    {
        cout << "RECOGNIZED '" << fromLanguage << "' " << result->Text << " (text could not be
translated)" << std::endl;
    }
    else if (result->Reason == ResultReason::NoMatch)
    {
        cout << "NOMATCH: Speech could not be recognized." << std::endl;
    }
    else if (result->Reason == ResultReason::Canceled)
    {
        auto cancellation = CancellationDetails::FromResult(result);
        cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

        if (cancellation->Reason == CancellationReason::Error)
        {
            cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
            cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
            cout << "CANCELED: Did you update the subscription info?" << std::endl;
        }
    }
}

int wmain()
{
    TranslateSpeechToText();
    return 0;
}

```

3. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
4. Replace the string `YourServiceRegion` with the `region` associated with your subscription (for example, `westus` for the free trial subscription).
5. From the menu bar, choose **File** > **Save All**.

Build and run the application

1. From the menu bar, select **Build > Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug > Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the text back to the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
```

Next steps

[Explore C++ samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to text in another language. After satisfying a few prerequisites, translating speech-to-text only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Update the `SpeechConfig` object to specify the source and target languages.
- Create a `TranslationRecognizer` object using the `SpeechConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

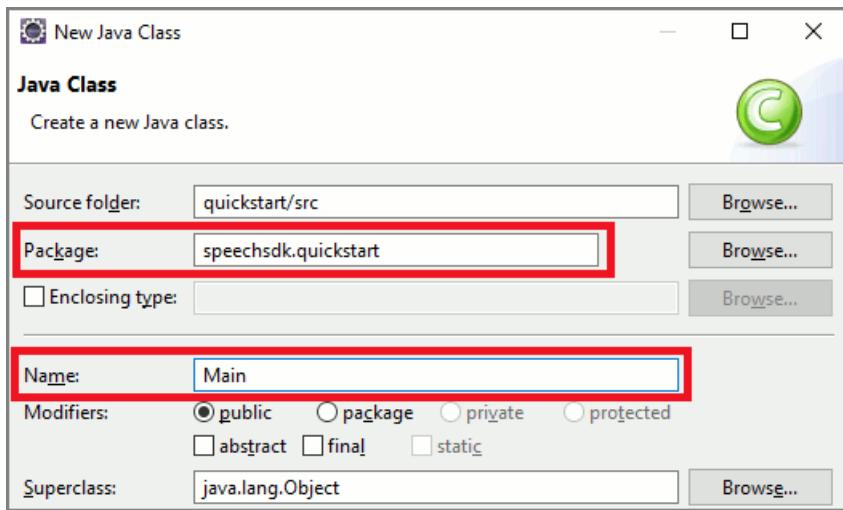
Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in `Main.java` with the following snippet:

```
package quickstart;

import java.io.IOException;
import java.util.concurrent.Future;
import java.util.concurrent.ExecutionException;
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.translation.*;

public class Main {

    public static void translationWithMicrophoneAsync() throws InterruptedException,
ExecutionException, IOException
    {
        // Creates an instance of a speech translation config with specified
        // subscription key and service region. Replace with your own subscription key
        // and service region (e.g., "westus").

        int exitCode = 1;
        SpeechTranslationConfig config =
SpeechTranslationConfig.fromSubscription("YourSubscriptionKey", "YourServiceRegion");
        assert(config != null);

        // Sets source and target languages.
        String fromLanguage = "en-US";
        String toLanguage = "de";
        config.setSpeechRecognitionLanguage(fromLanguage);
        config.addTargetLanguage(toLanguage);

        // Creates a translation recognizer using the default microphone audio input device.
        TranslationRecognizer recognizer = new TranslationRecognizer(config);
        assert(recognizer != null);

        System.out.println("Say something...");

        // Starts translation, and returns after a single utterance is recognized. The end of a
        // single utterance is determined by listening for silence at the end or until a maximum of
15
        // seconds of audio is processed. The task returns the recognized text as well as the
        // translation.
        // Note: Since recognizeOnceAsync() returns only a single utterance, it is suitable only for
single
        // shot recognition like command or query.
        // For long-running multi-utterance recognition, use startContinuousRecognitionAsync()
instead.
        Future<TranslationRecognitionResult> task = recognizer.recognizeOnceAsync();
        assert(task != null);

        TranslationRecognitionResult result = task.get();
```

```

        assert(result != null);

        if (result.getReason() == ResultReason.TranslatedSpeech) {
            System.out.println("RECOGNIZED '" + fromLanguage + "': " + result.getText());
            System.out.println("TRANSLATED into '" + toLanguage + "'": " +
result.getTranslations().get(toLanguage));
            exitCode = 0;
        }
        else if (result.getReason() == ResultReason.RecognizedSpeech) {
            System.out.println("RECOGNIZED '" + fromLanguage + "': " + result.getText() + "(text
could not be translated)");
            exitCode = 0;
        }
        else if (result.getReason() == ResultReason.NoMatch) {
            System.out.println("NOMATCH: Speech could not be recognized.");
        }
        else if (result.getReason() == ResultReason.Canceled) {
            CancellationDetails cancellation = CancellationDetails.fromResult(result);
            System.out.println("CANCELED: Reason=" + cancellation.getReason());

            if (cancellation.getReason() == CancellationReason.Error) {
                System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
                System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
                System.out.println("CANCELED: Did you update the subscription info?");
            }
        }
    }

    recognizer.close();

    System.exit(exitCode);
}

public static void main(String[] args) {
    try {
        translationWithMicrophoneAsync();
    } catch (Exception ex) {
        System.out.println("Unexpected exception: " + ex.getMessage());
        assert(false);
        System.exit(1);
    }
}
}

```

4. Replace the string `YourSubscriptionKey` with your subscription key.
5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
6. Save changes to the project.

Build and run the app

Press F11, or select **Run > Debug**.

1. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the text back to the application for display.

```

Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?

```

Next steps

[Explore Java samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to text in another language. After satisfying a few prerequisites, translating speech-to-text only takes five steps:

- Create a `SpeechConfig` object from your subscription key and region.
- Update the `SpeechConfig` object to specify the source and target languages.
- Create a `TranslationRecognizer` object using the `SpeechConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open `quickstart.py`, and replace all the code in it with the following.

```

import azure.cognitiveservices.speech as speechsdk

speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"

def translate_speech_to_text():

    # Creates an instance of a speech translation config with specified subscription key and service
    region.
    # Replace with your own subscription key and service region (e.g., "westus").
    translation_config = speechsdk.translation.SpeechTranslationConfig(subscription=speech_key,
    region=service_region)

    # Sets source and target languages.
    # Replace with the languages of your choice, from list found here: https://aka.ms/speech/sttt-
    languages
    fromLanguage = 'en-US'
    toLanguage = 'de'
    translation_config.speech_recognition_language = fromLanguage
    translation_config.add_target_language(toLanguage)

    # Creates a translation recognizer using and audio file as input.
    recognizer = speechsdk.translation.TranslationRecognizer(translation_config=translation_config)

    # Starts translation, and returns after a single utterance is recognized. The end of a
    # single utterance is determined by listening for silence at the end or until a maximum of 15
    # seconds of audio is processed. It returns the recognized text as well as the translation.
    # Note: Since recognize_once() returns only a single utterance, it is suitable only for single
    # shot recognition like command or query.
    # For long-running multi-utterance recognition, use start_continuous_recognition() instead.
    print("Say something...")
    result = recognizer.recognize_once()

    # Check the result
    if result.reason == speechsdk.ResultReason.TranslatedSpeech:
        print("RECOGNIZED '{}'".format(fromLanguage, result.text))
        print("TRANSLATED into '{}'".format(toLanguage, result.translations['de']))
    elif result.reason == speechsdk.ResultReason.RecognizedSpeech:
        print("RECOGNIZED: {} (text could not be translated)".format(result.text))
    elif result.reason == speechsdk.ResultReason.NoMatch:
        print("NOMATCH: Speech could not be recognized: {}".format(result.no_match_details))
    elif result.reason == speechsdk.ResultReason.Canceled:
        print("CANCELED: Reason={}".format(result.cancellation_details.reason))
        if result.cancellation_details.reason == speechsdk.CancellationReason.Error:
            print("CANCELED: ErrorDetails={}".format(result.cancellation_details.error_details))

translate_speech_to_text()

```

2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Replace the string `YourServiceRegion` with the `region` associated with your subscription (for example, `westus` for the free trial subscription).
4. Save the changes you've made to `quickstart.py`.

Build and run your app

1. Run the sample from the console or in your IDE:

```
python quickstart.py
```

2. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the text back to

the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
```

Next steps

[Explore Python samples on GitHub](#)

View or download all [Speech SDK Samples on GitHub](#).

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	CODE SAMPLES
C++	Quickstarts, Samples
C#	.NET Framework, .NET Core, UWP, Unity, Xamarin
Java	Android, JRE
Javascript	Browser
Node.js	Windows, Linux, macOS
Objective-C	iOS, macOS
Python	Windows, Linux, macOS
Swift	iOS, macOS

Quickstart: Translate speech to multiple languages

12/4/2019 • 14 minutes to read • [Edit Online](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech to text in multiple languages only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the speech recognition source language.
- Update the `SpeechTranslationConfig` object to specify multiple translation target languages.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [.NET](#)
- [.NET Core](#)

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open **Program.cs**, and replace all the code in it with the following.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Translation;

namespace helloworld
{
    class Program
    {
        public static async Task TranslateSpeechToText()
        {
            // Creates an instance of a speech translation config with specified subscription key and
            service region.
            // Replace with your own subscription key and service region (e.g., "westus").
            var config = SpeechTranslationConfig.FromSubscription("YourSubscriptionKey",
            "YourServiceRegion");

            // Sets source and target languages.
            // Replace with the languages of your choice, from list found here:
            https://aka.ms/speech/sttt-languages
            string fromLanguage = "en-US";
```

```

        config.SpeechRecognitionLanguage = fromLanguage;
        config.AddTargetLanguage("de");
        config.AddTargetLanguage("fr");

        // Creates a translation recognizer using the default microphone audio input device.
        using (var recognizer = new TranslationRecognizer(config))
        {
            // Starts translation, and returns after a single utterance is recognized. The end of a
            // single utterance is determined by listening for silence at the end or until a maximum
            of 15
            // seconds of audio is processed. The task returns the recognized text as well as the
            translation.
            // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only
            for single
            // shot recognition like command or query.
            // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync()
            instead.

            Console.WriteLine("Say something...");
            var result = await recognizer.RecognizeOnceAsync();

            // Checks result.
            if (result.Reason == ResultReason.TranslatedSpeech)
            {
                Console.WriteLine($"RECOGNIZED '{fromLanguage}': {result.Text}");
                foreach (var element in result.Translations)
                {
                    Console.WriteLine($"TRANSLATED into '{element.Key}': {element.Value}");
                }
            }
            else if (result.Reason == ResultReason.RecognizedSpeech)
            {
                Console.WriteLine($"RECOGNIZED '{fromLanguage}': {result.Text} (text could not be
translated)");
            }
            else if (result.Reason == ResultReason.NoMatch)
            {
                Console.WriteLine($"NOMATCH: Speech could not be recognized.");
            }
            else if (result.Reason == ResultReason.Canceled)
            {
                var cancellation = CancellationDetails.FromResult(result);
                Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

                if (cancellation.Reason == CancellationReason.Error)
                {
                    Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
                    Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
                    Console.WriteLine($"CANCELED: Did you update the subscription info?");
                }
            }
        }

        static void Main(string[] args)
        {
            TranslateSpeechToText().Wait();
        }
    }
}

```

2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
4. From the menu bar, choose **File > Save All**.

Build and run the application

1. From the menu bar, select **Build > Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug > Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to French and German). The Speech service then sends the text back to the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
TRANSLATED into 'fr': Quel temps fait-il à Seattle ?
```

Next steps

[Explore C# samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech to text in multiple languages only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the speech recognition source language.
- Update the `SpeechTranslationConfig` object to specify multiple translation target languages.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open the source file **helloworld.cpp**.
2. Replace all the code with the following snippet:

```
#include <iostream>
#include <vector>
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Translation;
```

```

void TranslateSpeechToText()
{
    // Creates an instance of a speech translation config with specified subscription key and service
    // region.
    // Replace with your own subscription key and service region (e.g., "westus").
    auto config = SpeechTranslationConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");

    // Sets source and target languages.
    // Replace with the languages of your choice, from list found here: https://aka.ms/speech/sttt-
    languages
    auto fromLanguage = "en-US";
    config->SetSpeechRecognitionLanguage(fromLanguage);
    config->AddTargetLanguage("de");
    config->AddTargetLanguage("fr");

    // Creates a translation recognizer using the default microphone audio input device.
    auto recognizer = TranslationRecognizer::FromConfig(config);

    // Starts translation, and returns after a single utterance is recognized. The end of a
    // single utterance is determined by listening for silence at the end or until a maximum of 15
    // seconds of audio is processed. The task returns the recognized text as well as the translation.
    // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
    // shot recognition like command or query.
    // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
    cout << "Say something...\n";
    auto result = recognizer->RecognizeOnceAsync().get();

    // Checks result.
    if (result->Reason == ResultReason::TranslatedSpeech)
    {
        cout << "RECOGNIZED '" << fromLanguage << "'": " << result->Text << std::endl;
        for (const auto& it : result->Translations)
        {
            cout << "TRANSLATED into '" << it.first.c_str() << "'": " << it.second.c_str() << std::endl;
        }
    }
    else if (result->Reason == ResultReason::RecognizedSpeech)
    {
        cout << "RECOGNIZED '" << fromLanguage << "' " << result->Text << " (text could not be
        translated)" << std::endl;
    }
    else if (result->Reason == ResultReason::NoMatch)
    {
        cout << "NOMATCH: Speech could not be recognized." << std::endl;
    }
    else if (result->Reason == ResultReason::Canceled)
    {
        auto cancellation = CancellationDetails::FromResult(result);
        cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

        if (cancellation->Reason == CancellationReason::Error)
        {
            cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
            cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
            cout << "CANCELED: Did you update the subscription info?" << std::endl;
        }
    }
}

int wmain()
{
    TranslateSpeechToText();
    return 0;
}

```

3. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
4. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example,

[westus](#) for the free trial subscription).

5. From the menu bar, choose **File > Save All**.

Build and run the application

1. From the menu bar, select **Build > Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug > Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to French and German). The Speech service then sends the text back to the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
TRANSLATED into 'fr': Quel temps fait-il à Seattle ?
```

Next steps

[Explore C++ samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech to text in multiple languages only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the speech recognition source language.
- Update the `SpeechTranslationConfig` object to specify multiple translation target languages.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

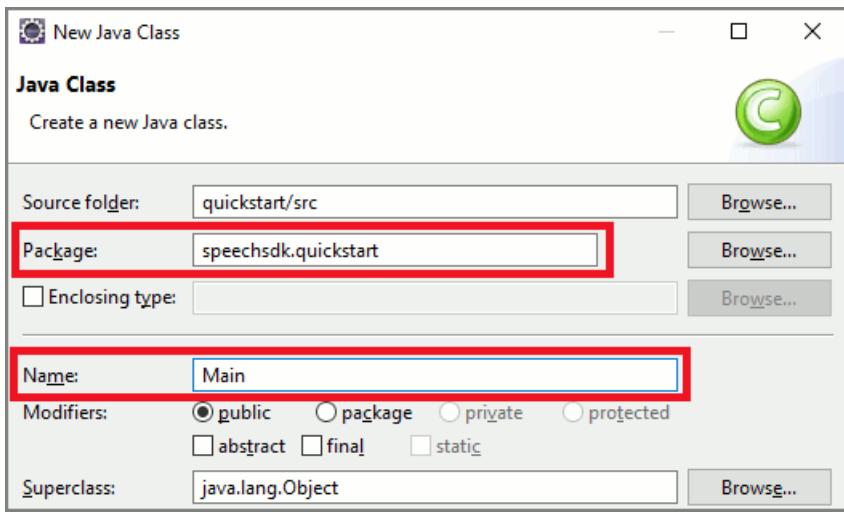
Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in `Main.java` with the following snippet:

```
package quickstart;

import java.io.IOException;
import java.util.concurrent.Future;
import java.util.Map;
import java.util.concurrent.ExecutionException;
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.translation.*;

public class Main {

    public static void translationWithMicrophoneAsync() throws InterruptedException, ExecutionException,
    IOException
    {
        // Creates an instance of a speech translation config with specified
        // subscription key and service region. Replace with your own subscription key
        // and service region (e.g., "westus").

        int exitCode = 1;
        SpeechTranslationConfig config = SpeechTranslationConfig.fromSubscription("YourSubscriptionKey",
        "YourServiceRegion");
        assert(config != null);

        // Sets source and target languages.
        String fromLanguage = "en-US";
        config.setSpeechRecognitionLanguage(fromLanguage);
        config.addTargetLanguage("de");
        config.addTargetLanguage("fr");

        // Creates a translation recognizer using the default microphone audio input device.
        TranslationRecognizer recognizer = new TranslationRecognizer(config);
        assert(recognizer != null);

        System.out.println("Say something...");

        // Starts translation, and returns after a single utterance is recognized. The end of a
        // single utterance is determined by listening for silence at the end or until a maximum of 15
        // seconds of audio is processed. The task returns the recognized text as well as the
        translation.
        // Note: Since recognizeOnceAsync() returns only a single utterance, it is suitable only for
        single
        // shot recognition like command or query.
        // For long-running multi-utterance recognition, use startContinuousRecognitionAsync() instead.
        Future<TranslationRecognitionResult> task = recognizer.recognizeOnceAsync();
        assert(task != null);

        TranslationRecognitionResult result = task.get();
        assert(result != null);
    }
}
```

```

        if (result.getReason() == ResultReason.TranslatedSpeech) {
            System.out.println("RECOGNIZED '" + fromLanguage + "': " + result.getText());

            Map<String, String> map = result.getTranslations();
            for(String toLanguage: map.keySet()) {
                System.out.println("TRANSLATED into '" + toLanguage + "': " + map.get(toLanguage));
            }
            exitCode = 0;
        }
        else if (result.getReason() == ResultReason.RecognizedSpeech) {
            System.out.println("RECOGNIZED '" + fromLanguage + "': " + result.getText() + "(text could
not be translated)");
            exitCode = 0;
        }
        else if (result.getReason() == ResultReason.NoMatch) {
            System.out.println("NOMATCH: Speech could not be recognized.");
        }
        else if (result.getReason() == ResultReason.Canceled) {
            CancellationDetails cancellation = CancellationDetails.fromResult(result);
            System.out.println("CANCELED: Reason=" + cancellation.getReason());

            if (cancellation.getReason() == CancellationReason.Error) {
                System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
                System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
                System.out.println("CANCELED: Did you update the subscription info?");
            }
        }
    }

    recognizer.close();

    System.exit(exitCode);
}

public static void main(String[] args) {
    try {
        translationWithMicrophoneAsync();
    } catch (Exception ex) {
        System.out.println("Unexpected exception: " + ex.getMessage());
        assert(false);
        System.exit(1);
    }
}

```

4. Replace the string `YourSubscriptionKey` with your subscription key.
5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
6. Save changes to the project.

Build and run the app

Press F11, or select **Run > Debug**.

1. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to French and German). The Speech service then sends the text back to the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
TRANSLATED into 'fr': Quel temps fait-il à Seattle ?
```

Next steps

[Explore Java samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech to text in multiple languages only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the speech recognition source language.
- Update the `SpeechTranslationConfig` object to specify multiple translation target languages.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open `quickstart.py`, and replace all the code in it with the following.

```

import azure.cognitiveservices.speech as speechsdk

speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"

def translate_speech_to_text():

    # Creates an instance of a speech translation config with specified subscription key and service
    region.
    # Replace with your own subscription key and service region (e.g., "westus").
    translation_config = speechsdk.translation.SpeechTranslationConfig(subscription=speech_key,
    region=service_region)

    # Sets source and target languages.
    # Replace with the languages of your choice, from list found here: https://aka.ms/speech/sttt-
    languages
    fromLanguage = 'en-US'
    translation_config.speech_recognition_language = fromLanguage
    translation_config.add_target_language('de')
    translation_config.add_target_language('fr')

    # Creates a translation recognizer using an audio file as input.
    recognizer = speechsdk.translation.TranslationRecognizer(translation_config=translation_config)

    # Starts translation, and returns after a single utterance is recognized. The end of a
    # single utterance is determined by listening for silence at the end or until a maximum of 15
    # seconds of audio is processed. It returns the recognized text as well as the translation.
    # Note: Since recognize_once() returns only a single utterance, it is suitable only for single
    # shot recognition like command or query.
    # For long-running multi-utterance recognition, use start_continuous_recognition() instead.
    print("Say something...")
    result = recognizer.recognize_once()

    # Check the result
    if result.reason == speechsdk.ResultReason.TranslatedSpeech:
        print("RECOGNIZED '{}'".format(fromLanguage, result.text))
        print("TRANSLATED into {}: {}".format('de', result.translations['de']))
        print("TRANSLATED into {}: {}".format('fr', result.translations['fr']))
    elif result.reason == speechsdk.ResultReason.RecognizedSpeech:
        print("RECOGNIZED: {} (text could not be translated)".format(result.text))
    elif result.reason == speechsdk.ResultReason.NoMatch:
        print("NOMATCH: Speech could not be recognized: {}".format(result.no_match_details))
    elif result.reason == speechsdk.ResultReason.Canceled:
        print("CANCELED: Reason={}".format(result.cancellation_details.reason))
        if result.cancellation_details.reason == speechsdk.CancellationReason.Error:
            print("CANCELED: ErrorDetails={}".format(result.cancellation_details.error_details))

translate_speech_to_text()

```

2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
4. Save the changes you've made to `quickstart.py`.

Build and run your app

1. Run the sample from the console or in your IDE:

```
python quickstart.py
```

2. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to French and German). The Speech service then sends the

text back to the application for display.

```
Say something...
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
TRANSLATED into 'fr': Quel temps fait-il à Seattle ?
```

Next steps

[Explore Python samples on GitHub](#)

View or download all [Speech SDK Samples](#) on GitHub.

Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

LANGUAGE	CODE SAMPLES
C++	Quickstarts , Samples
C#	.NET Framework , .NET Core , UWP , Unity , Xamarin
Java	Android , JRE
Javascript	Browser
Node.js	Windows , Linux , macOS
Objective-C	iOS , macOS
Python	Windows , Linux , macOS
Swift	iOS , macOS

Quickstart: Translate speech-to-speech

12/4/2019 • 15 minutes to read • [Edit Online](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech-to-speech only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the source and target languages.
- Update the `SpeechTranslationConfig` object to specify the speech output voice name.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C# Samples](#) on GitHub. Otherwise, let's get started.

Choose your target environment

- [.NET](#)
- [.NET Core](#)

Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

Add sample code

1. Open **Program.cs**, and replace all the code in it with the following.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Translation;

namespace helloworld
{
    class Program
    {
        public static async Task TranslateSpeechToSpeech()
        {
            // Creates an instance of a speech translation config with specified subscription key and
            service region.
            // Replace with your own subscription key and service region (e.g., "westus").
            var config = SpeechTranslationConfig.FromSubscription("YourSubscriptionKey",
            "YourServiceRegion");

            // Sets source and target languages.
            // Replace with the languages of your choice, from list found here:
            https://aka.ms/speech/sttt-languages
            string fromLanguage = "en-US";
            string toLanguage = "de";
```

```

config.SpeechRecognitionLanguage = fromLanguage;
config.AddTargetLanguage(toLanguage);

// Sets the synthesis output voice name.
// Replace with the languages of your choice, from list found here:
https://aka.ms/speech/tts-languages
config.VoiceName = "de-DE-Hedda";

// Creates a translation recognizer using the default microphone audio input device.
using (var recognizer = new TranslationRecognizer(config))
{
    // Prepare to handle the synthesized audio data.
    recognizer.Synthesizing += (s, e) =>
    {
        var audio = e.Result.GetAudio();
        Console.WriteLine(audio.Length != 0
            ? $"AUDIO SYNTHESIZED: {audio.Length} byte(s)"
            : $"AUDIO SYNTHESIZED: {audio.Length} byte(s) (COMPLETE)");
    };

    // Starts translation, and returns after a single utterance is recognized. The end of a
    // single utterance is determined by listening for silence at the end or until a maximum
of 15
    // seconds of audio is processed. The task returns the recognized text as well as the
translation.
    // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only
for single
    // shot recognition like command or query.
    // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync()
instead.
    Console.WriteLine("Say something...");
    var result = await recognizer.RecognizeOnceAsync();

    // Checks result.
    if (result.Reason == ResultReason.TranslatedSpeech)
    {
        Console.WriteLine($"RECOGNIZED '{fromLanguage}': {result.Text}");
        Console.WriteLine($"TRANSLATED into '{toLanguage}': {result.Translations[toLanguage]}");
    }
    else if (result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"RECOGNIZED '{fromLanguage}': {result.Text} (text could not be
translated)");
    }
    else if (result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
    else if (result.Reason == ResultReason.Canceled)
    {
        var cancellation = CancellationDetails.FromResult(result);
        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

        if (cancellation.Reason == CancellationReason.Error)
        {
            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
            Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
            Console.WriteLine($"CANCELED: Did you update the subscription info?");
        }
    }
}

static void Main(string[] args)
{
    TranslateSpeechToSpeech().Wait();
}
}

```

- ```
}
```
2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
  3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
  4. From the menu bar, choose **File** > **Save All**.

## Build and run the application

1. From the menu bar, select **Build** > **Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug** > **Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the synthesized audio and the text back to the application for display.

```
Say something...
AUDIO SYNTHESIZED: 76784 byte(s)
AUDIO SYNTHESIZED: 0 byte(s)(COMPLETE)
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
```

## Next steps

[Explore C# samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech-to-speech only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the source and target languages.
- Update the `SpeechTranslationConfig` object to specify the speech output voice name.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK C++ Samples](#) on GitHub. Otherwise, let's get started.

## Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

## Add sample code

1. Open the source file **helloworld.cpp**.

2. Replace all the code with the following snippet:

```
#include <iostream>
#include <vector>
#include <speechapi_cxx.h>

using namespace std;
using namespace Microsoft::CognitiveServices::Speech;
using namespace Microsoft::CognitiveServices::Speech::Translation;

void TranslateSpeechToSpeech()
{
 // Creates an instance of a speech translation config with specified subscription key and service
 region.
 // Replace with your own subscription key and service region (e.g., "westus").
 auto config = SpeechTranslationConfig::FromSubscription("YourSubscriptionKey", "YourServiceRegion");

 // Sets source and target languages.
 // Replace with the languages of your choice, from list found here: https://aka.ms/speech/sttt-
 languages
 auto fromLanguage = "en-US";
 auto toLanguage = "de";
 config->SetSpeechRecognitionLanguage(fromLanguage);
 config->AddTargetLanguage(toLanguage);

 // Sets the synthesis output voice name.
 // Replace with the languages of your choice, from list found here: https://aka.ms/speech/tts-
 languages
 config->SetVoiceName("de-DE-Hedda");

 // Creates a translation recognizer using the default microphone audio input device.
 auto recognizer = TranslationRecognizer::FromConfig(config);

 // Prepare to handle the synthesized audio data.
 recognizer->Synthesizing.Connect([](const TranslationSynthesisEventArgs& e)
 {
 auto size = e.Result->Audio.size();
 cout << "AUDIO SYNTHESIZED: " << size << " byte(s)" << (size == 0 ? "(COMPLETE)" : "") <<
 std::endl;
 });

 // Starts translation, and returns after a single utterance is recognized. The end of a
 // single utterance is determined by listening for silence at the end or until a maximum of 15
 // seconds of audio is processed. The task returns the recognized text as well as the translation.
 // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
 // shot recognition like command or query.
 // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
 cout << "Say something...\n";
 auto result = recognizer->RecognizeOnceAsync().get();

 // Checks result.
 if (result->Reason == ResultReason::TranslatedSpeech)
 {
 cout << "RECOGNIZED '" << fromLanguage << "' : " << result->Text << std::endl;
 cout << "TRANSLATED into '" << toLanguage << "' : " << result->Translations.at(toLanguage) <<
 std::endl;
 }
 else if (result->Reason == ResultReason::RecognizedSpeech)
 {
 cout << "RECOGNIZED '" << fromLanguage << "' " << result->Text << " (text could not be
 translated)" << std::endl;
 }
 else if (result->Reason == ResultReason::NoMatch)
 {
 cout << "NOMATCH: Speech could not be recognized." << std::endl;
 }
 else if (result->Reason == ResultReason::Canceled)
 {
```

```

 auto cancellation = CancellationDetails::FromResult(result);
 cout << "CANCELED: Reason=" << (int)cancellation->Reason << std::endl;

 if (cancellation->Reason == CancellationReason::Error)
 {
 cout << "CANCELED: ErrorCode=" << (int)cancellation->ErrorCode << std::endl;
 cout << "CANCELED: ErrorDetails=" << cancellation->ErrorDetails << std::endl;
 cout << "CANCELED: Did you update the subscription info?" << std::endl;
 }
 }

 int wmain()
 {
 TranslateSpeechToSpeech();
 return 0;
 }
}

```

3. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
4. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
5. From the menu bar, choose **File** > **Save All**.

## Build and run the application

1. From the menu bar, select **Build** > **Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug** > **Start Debugging** (or press **F5**) to start the **helloworld** application.
3. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the synthesized audio and the text back to the application for display.

```

Say something...
AUDIO SYNTHESIZED: 76784 byte(s)
AUDIO SYNTHESIZED: 0 byte(s)(COMPLETE)
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?

```

## Next steps

[Explore C++ samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech-to-speech only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the source and target languages.
- Update the `SpeechTranslationConfig` object to specify the speech output voice name.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Java Samples](#) on GitHub. Otherwise, let's get started.

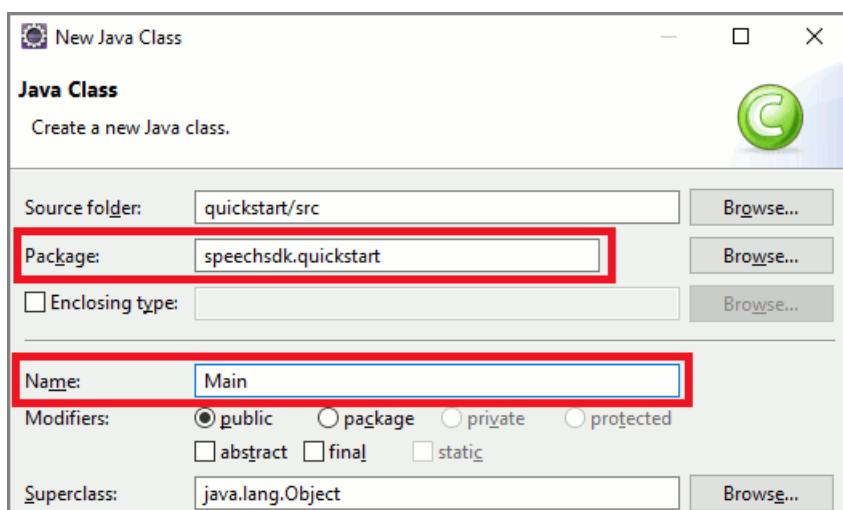
# Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

## Add sample code

1. To add a new empty class to your Java project, select **File > New > Class**.
2. In the **New Java Class** window, enter **speechsdk.quickstart** into the **Package** field, and **Main** into the **Name** field.



3. Replace all code in **Main.java** with the following snippet:

```
package quickstart;

import java.io.IOException;
import java.util.concurrent.Future;
import java.util.concurrent.ExecutionException;
import com.microsoft.cognitiveservices.speech.*;
import com.microsoft.cognitiveservices.speech.translation.*;

public class Main {

 public static void translateSpeechToSpeech() throws InterruptedException, ExecutionException,
 IOException
 {
 // Creates an instance of a speech translation config with specified
 // subscription key and service region. Replace with your own subscription key
 // and service region (e.g., "westus").

 int exitCode = 1;
 SpeechTranslationConfig config = SpeechTranslationConfig.fromSubscription("YourSubscriptionKey",
 "YourServiceRegion");
 assert(config != null);

 // Sets source and target languages.
 String fromLanguage = "en-US";
 String toLanguage = "de";
 config.setSpeechRecognitionLanguage(fromLanguage);
 config.addTargetLanguage(toLanguage);

 // Sets the synthesis output voice name.
 }
}
```

```

// Replace with the languages of your choice, from list found here: https://aka.ms/speech/tts-languages
config.setVoiceName("de-DE-Hedda");

// Creates a translation recognizer using the default microphone audio input device.
TranslationRecognizer recognizer = new TranslationRecognizer(config);
assert(recognizer != null);

// Prepare to handle the synthesized audio data.
recognizer.synthesizing.addEventListerner((s, e) -> {
 int size = e.getResult().getAudio().length;
 System.out.println(size != 0
 ? "AUDIO SYNTHESIZED: " + size + " byte(s)"
 : "AUDIO SYNTHESIZED: " + size + " byte(s) (COMPLETE)");
});

System.out.println("Say something...");

// Starts translation, and returns after a single utterance is recognized. The end of a
// single utterance is determined by listening for silence at the end or until a maximum of 15
// seconds of audio is processed. The task returns the recognized text as well as the
translation.
// Note: Since recognizeOnceAsync() returns only a single utterance, it is suitable only for
single

// shot recognition like command or query.
// For long-running multi-utterance recognition, use startContinuousRecognitionAsync() instead.
Future<TranslationRecognitionResult> task = recognizer.recognizeOnceAsync();
assert(task != null);

TranslationRecognitionResult result = task.get();
assert(result != null);

if (result.getReason() == ResultReason.TranslatedSpeech) {
 System.out.println("RECOGNIZED '" + fromLanguage + "': " + result.getText());
 System.out.println("TRANSLATED into '" + toLanguage + "': " +
result.getTranslations().get(toLanguage));
 exitCode = 0;
}
else if (result.getReason() == ResultReason.RecognizedSpeech) {
 System.out.println("RECOGNIZED '" + fromLanguage + "': " + result.getText() + "(text could
not be translated)");
 exitCode = 0;
}
else if (result.getReason() == ResultReason.NoMatch) {
 System.out.println("NOMATCH: Speech could not be recognized.");
}
else if (result.getReason() == ResultReason.Canceled) {
 CancellationDetails cancellation = CancellationDetails.fromResult(result);
 System.out.println("CANCELED: Reason=" + cancellation.getReason());

 if (cancellation.getReason() == CancellationReason.Error) {
 System.out.println("CANCELED: ErrorCode=" + cancellation.getErrorCode());
 System.out.println("CANCELED: ErrorDetails=" + cancellation.getErrorDetails());
 System.out.println("CANCELED: Did you update the subscription info?");
 }
}

recognizer.close();

System.exit(exitCode);
}

public static void main(String[] args) {
try {
 translateSpeechToSpeech();
} catch (Exception ex) {
 System.out.println("Unexpected exception: " + ex.getMessage());
 assert(false);
 System.exit(1);
}

```

```
 }
}
```

4. Replace the string `YourSubscriptionKey` with your subscription key.
5. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
6. Save changes to the project.

## Build and run the app

Press F11, or select **Run > Debug**.

1. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the synthesized audio and the text back to the application for display.

```
Say something...
AUDIO SYNTHESIZED: 76784 byte(s)
AUDIO SYNTHESIZED: 0 byte(s)(COMPLETE)
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
```

## Next steps

[Explore Java samples on GitHub](#)

In this quickstart you will use the [Speech SDK](#) to interactively translate speech from one language to speech in another language. After satisfying a few prerequisites, translating speech-to-speech only takes six steps:

- Create a `SpeechTranslationConfig` object from your subscription key and region.
- Update the `SpeechTranslationConfig` object to specify the source and target languages.
- Update the `SpeechTranslationConfig` object to specify the speech output voice name.
- Create a `TranslationRecognizer` object using the `SpeechTranslationConfig` object from above.
- Using the `TranslationRecognizer` object, start the recognition process for a single utterance.
- Inspect the `TranslationRecognitionResult` returned.

If you prefer to jump right in, view or download all [Speech SDK Python Samples](#) on GitHub. Otherwise, let's get started.

## Prerequisites

Before you get started, make sure to:

- [Create an Azure Speech Resource](#)
- [Setup your development environment](#)
- [Create an empty sample project](#)

## Add sample code

1. Open `quickstart.py`, and replace all the code in it with the following.

```

import azure.cognitiveservices.speech as speechsdk

speech_key, service_region = "YourSubscriptionKey", "YourServiceRegion"

def translate_speech_to_speech():

 # Creates an instance of a speech translation config with specified subscription key and service
 region.
 # Replace with your own subscription key and service region (e.g., "westus").
 translation_config = speechsdk.translation.SpeechTranslationConfig(subscription=speech_key,
 region=service_region)

 # Sets source and target languages.
 # Replace with the languages of your choice, from list found here: https://aka.ms/speech/sttt-
 languages
 fromLanguage = 'en-US'
 toLanguage = 'de'
 translation_config.speech_recognition_language = fromLanguage
 translation_config.add_target_language(toLanguage)

 # Sets the synthesis output voice name.
 # Replace with the languages of your choice, from list found here: https://aka.ms/speech/tts-
 languages
 translation_config.voice_name = "de-DE-Hedda"

 # Creates a translation recognizer using and audio file as input.
 recognizer = speechsdk.translation.TranslationRecognizer(translation_config=translation_config)

 # Prepare to handle the synthesized audio data.
 def synthesis_callback(evt):
 size = len(evt.result.audio)
 print('AUDIO SYNTHESIZED: {} byte(s) {}'.format(size, '(COMPLETED)' if size == 0 else ''))

 recognizer.synthesizing.connect(synthesis_callback)

 # Starts translation, and returns after a single utterance is recognized. The end of a
 # single utterance is determined by listening for silence at the end or until a maximum of 15
 # seconds of audio is processed. It returns the recognized text as well as the translation.
 # Note: Since recognize_once() returns only a single utterance, it is suitable only for single
 # shot recognition like command or query.
 # For long-running multi-utterance recognition, use start_continuous_recognition() instead.
 print("Say something...")
 result = recognizer.recognize_once()

 # Check the result
 if result.reason == speechsdk.ResultReason.TranslatedSpeech:
 print("RECOGNIZED '{}': {}".format(fromLanguage, result.text))
 print("TRANSLATED into {}: {}".format(toLanguage, result.translations['de']))
 elif result.reason == speechsdk.ResultReason.RecognizedSpeech:
 print("RECOGNIZED: {} (text could not be translated)".format(result.text))
 elif result.reason == speechsdk.ResultReason.NoMatch:
 print("NOMATCH: Speech could not be recognized: {}".format(result.no_match_details))
 elif result.reason == speechsdk.ResultReason.Canceled:
 print("CANCELED: Reason={}".format(result.cancellation_details.reason))
 if result.cancellation_details.reason == speechsdk.CancellationReason.Error:
 print("CANCELED: ErrorDetails={}".format(result.cancellation_details.error_details))

translate_speech_to_speech()

```

2. In the same file, replace the string `YourSubscriptionKey` with your subscription key.
3. Replace the string `YourServiceRegion` with the [region](#) associated with your subscription (for example, `westus` for the free trial subscription).
4. Save the changes you've made to `quickstart.py`.

# Build and run your app

1. Run the sample from the console or in your IDE:

```
python quickstart.py
```

2. Speak an English phrase or sentence. The application transmits your speech to the Speech service, which translates and transcribes to text (in this case, to German). The Speech service then sends the synthesized audio and the text back to the application for display.

```
Say something...
AUDIO SYNTHESIZED: 76784 byte(s)
AUDIO SYNTHESIZED: 0 byte(s) (COMPLETE)
RECOGNIZED 'en-US': What's the weather in Seattle?
TRANSLATED into 'de': Wie ist das Wetter in Seattle?
```

## Next steps

[Explore Python samples on GitHub](#)

View or download all [Speech SDK Samples](#) on GitHub.

## Additional language and platform support

If you've clicked this tab, you probably didn't see a quickstart in your favorite programming language. Don't worry, we have additional quickstart materials and code samples available on GitHub. Use the table to find the right sample for your programming language and platform/OS combination.

| LANGUAGE    | CODE SAMPLES                                                   |
|-------------|----------------------------------------------------------------|
| C++         | <a href="#">Quickstarts, Samples</a>                           |
| C#          | <a href="#">.NET Framework, .NET Core, UWP, Unity, Xamarin</a> |
| Java        | <a href="#">Android, JRE</a>                                   |
| Javascript  | <a href="#">Browser</a>                                        |
| Node.js     | <a href="#">Windows, Linux, macOS</a>                          |
| Objective-C | <a href="#">iOS, macOS</a>                                     |
| Python      | <a href="#">Windows, Linux, macOS</a>                          |
| Swift       | <a href="#">iOS, macOS</a>                                     |

# Tutorial: Build a Flask app with Azure Cognitive Services

12/10/2019 • 24 minutes to read • [Edit Online](#)

In this tutorial, you'll build a Flask web app that uses Azure Cognitive Services to translate text, analyze sentiment, and synthesize translated text into speech. Our focus is on the Python code and Flask routes that enable our application, however, we will help you out with the HTML and Javascript that pulls the app together. If you run into any issues let us know using the feedback button below.

Here's what this tutorial covers:

- Get Azure subscription keys
- Set up your development environment and install dependencies
- Create a Flask app
- Use the Translator Text API to translate text
- Use Text Analytics to analyze positive/negative sentiment of input text and translations
- Use Speech Services to convert translated text into synthesized speech
- Run your Flask app locally

## TIP

If you'd like to skip ahead and see all the code at once, the entire sample, along with build instructions are available on [GitHub](#).

## What is Flask?

Flask is a microframework for creating web applications. This means Flask provides you with tools, libraries, and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as substantive as a web-based calendar application or a commercial website.

For those of you who want to deep dive after this tutorial here are a few helpful links:

- [Flask documentation](#)
- [Flask for Dummies - A Beginner's Guide to Flask](#)

## Prerequisites

Let's review the software and subscription keys that you'll need for this tutorial.

- [Python 3.5.2 or later](#)
- [Git tools](#)
- An IDE or text editor, such as [Visual Studio Code](#) or [Atom](#)
- [Chrome](#) or [Firefox](#)
- A **Translator Text** subscription key (Note that you aren't required to select a region.)
- A **Text Analytics** subscription key in the **West US** region.
- A **Speech Services** subscription key in the **West US** region.

## Create an account and subscribe to resources

As previously mentioned, you're going to need three subscription keys for this tutorial. This means that you need to create a resource within your Azure account for:

- Translator Text
- Text Analytics
- Speech Services

Use [Create a Cognitive Services Account in the Azure portal](#) for step-by-step instructions to create resources.

#### IMPORTANT

For this tutorial, please create your resources in the West US region. If using a different region, you'll need to adjust the base URL in each of your Python files.

## Set up your dev environment

Before you build your Flask web app, you'll need to create a working directory for your project and install a few Python packages.

### Create a working directory

1. Open command line (Windows) or terminal (macOS/Linux). Then, create a working directory and sub directories for your project:

```
mkdir -p flask-cog-services/static/scripts && mkdir flask-cog-services/templates
```

2. Change to your project's working directory:

```
cd flask-cog-services
```

### Create and activate your virtual environment with `virtualenv`

Let's create a virtual environment for our Flask app using `virtualenv`. Using a virtual environment ensures that you have a clean environment to work from.

1. In your working directory, run this command to create a virtual environment: **macOS/Linux:**

```
virtualenv venv --python=python3
```

We've explicitly declared that the virtual environment should use Python 3. This ensures that users with multiple Python installations are using the correct version.

### Windows CMD / Windows Bash:

```
virtualenv venv
```

To keep things simple, we're naming your virtual environment `venv`.

2. The commands to activate your virtual environment will vary depending on your platform/shell:

| PLATFORM    | SHELL    | COMMAND                               |
|-------------|----------|---------------------------------------|
| macOS/Linux | bash/zsh | <code>source venv/bin/activate</code> |

| PLATFORM | SHELL        | COMMAND                                   |
|----------|--------------|-------------------------------------------|
| Windows  | bash         | <code>source venv/Scripts/activate</code> |
|          | Command Line | <code>venv\Scripts\activate.bat</code>    |
|          | PowerShell   | <code>venv\Scripts\Activate.ps1</code>    |

After running this command, your command line or terminal session should be prefaced with `venv`.

3. You can deactivate the session at any time by typing this into the command line or terminal: `deactivate`.

#### NOTE

Python has extensive documentation for creating and managing virtual environments, see [virtualenv](#).

### Install requests

Requests is a popular module that is used to send HTTP 1.1 requests. There's no need to manually add query strings to your URLs, or to form-encode your POST data.

1. To install requests, run:

```
pip install requests
```

#### NOTE

If you'd like to learn more about requests, see [Requests: HTTP for Humans](#).

### Install and configure Flask

Next we need to install Flask. Flask handles the routing for our web app, and allows us to make server-to-server calls that hide our subscription keys from the end user.

1. To install Flask, run:

```
pip install Flask
```

Let's make sure Flask was installed. Run:

```
flask --version
```

The version should be printed to terminal. Anything else means something went wrong.

2. To run the Flask app, you can either use the `flask` command or Python's `-m` switch with Flask. Before you can do that you need to tell your terminal which app to work with by exporting the `FLASK_APP` environment variable:

#### macOS/Linux:

```
export FLASK_APP=app.py
```

#### Windows:

```
set FLASK_APP=app.py
```

## Create your Flask app

In this section, you're going to create a barebones Flask app that returns an HTML file when users hit the root of your app. Don't spend too much time trying to pick apart the code, we'll come back to update this file later.

### What is a Flask route?

Let's take a minute to talk about "routes". Routing is used to bind a URL to a specific function. Flask uses route decorators to register functions to specific URLs. For example, when a user navigates to the root (/) of our web app, index.html is rendered.

```
@app.route('/')
def index():
 return render_template('index.html')
```

Let's take a look at one more example to hammer this home.

```
@app.route('/about')
def about():
 return render_template('about.html')
```

This code ensures that when a user navigates to <http://your-web-app.com/about> that the about.html file is rendered.

While these samples illustrate how to render html pages for a user, routes can also be used to call APIs when a button is pressed, or take any number of actions without having to navigate away from the homepage. You'll see this in action when you create routes for translation, sentiment, and speech synthesis.

### Get started

1. Open the project in your IDE, then create a file named `app.py` in the root of your working directory. Next, copy this code into `app.py` and save:

```
from flask import Flask, render_template, url_for, jsonify, request

app = Flask(__name__)
app.config['JSON_AS_ASCII'] = False

@app.route('/')
def index():
 return render_template('index.html')
```

This code block tells the app to display `index.html` whenever a user navigates to the root of your web app (/).

2. Next, let's create the front-end for our web app. Create a file named `index.html` in the `templates` directory. Then copy this code into `templates/index.html`.

```

<!doctype html>
<html lang="en">
 <head>
 <!-- Required metadata tags -->
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
 <meta name="description" content="Translate and analyze text with Azure Cognitive Services.">
 <!-- Bootstrap CSS -->
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
 <title>Translate and analyze text with Azure Cognitive Services</title>
 </head>
 <body>
 <div class="container">
 <h1>Translate, synthesize, and analyze text with Azure</h1>
 <p>This simple web app uses Azure for text translation, text-to-speech conversion, and sentiment analysis of input text and translations. Learn more about Azure Cognitive Services.</p>
 <!-- HTML provided in the following sections goes here. -->
 <!-- End -->
 </div>

 <!-- Required Javascript for this tutorial -->
 <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkVYIK3UENmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
 <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8j0t6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
 <script type="text/javascript" src="static/scripts/main.js"></script>
 </body>
</html>

```

3. Let's test the Flask app. From the terminal, run:

```
flask run
```

4. Open a browser and navigate to the URL provided. You should see your single page app. Press **Ctrl + c** to kill the app.

## Translate text

Now that you have an idea of how a simple Flask app works, let's:

- Write some Python to call the Translator Text API and return a response
- Create a Flask route to call your Python code
- Update the HTML with an area for text input and translation, a language selector, and translate button
- Write Javascript that allows users to interact with your Flask app from the HTML

### Call the Translator Text API

The first thing you need to do is write a function to call the Translator Text API. This function will take two arguments: `text_input` and `language_output`. This function is called whenever a user presses the translate button in your app. The text area in the HTML is sent as the `text_input`, and the language selection value in the HTML is sent as `language_output`.

1. Let's start by creating a file called `translate.py` in the root of your working directory.
2. Next, add this code to `translate.py`. This function takes two arguments: `text_input` and `language_output`.

```

import os, requests, uuid, json

Don't forget to replace with your Cog Services subscription key!
If you prefer to use environment variables, see Extra Credit for more info.
subscription_key = 'YOUR_TRANSLATOR_TEXT_SUBSCRIPTION_KEY'

Don't forget to replace with your Cog Services location!
Our Flask route will supply two arguments: text_input and language_output.
When the translate text button is pressed in our Flask app, the Ajax request
will grab these values from our web app, and use them in the request.
See main.js for Ajax calls.
def get_translation(text_input, language_output):
 base_url = 'https://api.cognitive.microsofttranslator.com'
 path = '/translate?api-version=3.0'
 params = '&to=' + language_output
 constructed_url = base_url + path + params

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

 # You can pass more than one object in body.
 body = [{{
 'text' : text_input
 }}]
 response = requests.post(constructed_url, headers=headers, json=body)
 return response.json()

```

3. Add your Translator Text subscription key and save.

#### Add a route to `app.py`

Next, you'll need to create a route in your Flask app that calls `translate.py`. This route will be called each time a user presses the translate button in your app.

For this app, your route is going to accept `POST` requests. This is because the function expects the text to translate and an output language for the translation.

Flask provides helper functions to help you parse and manage each request. In the code provided, `get_json()` returns the data from the `POST` request as JSON. Then using `data['text']` and `data['to']`, the text and output language values are passed to `get_translation()` function available from `translate.py`. The last step is to return the response as JSON, since you'll need to display this data in your web app.

In the following sections, you'll repeat this process as you create routes for sentiment analysis and speech synthesis.

1. Open `app.py` and locate the import statement at the top of `app.py` and add the following line:

```
import translate
```

Now our Flask app can use the method available via `translate.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/translate-text', methods=['POST'])
def translate_text():
 data = request.get_json()
 text_input = data['text']
 translation_output = data['to']
 response = translate.get_translation(text_input, translation_output)
 return jsonify(response)
```

## Update `index.html`

Now that you have a function to translate text, and a route in your Flask app to call it, the next step is to start building the HTML for your app. The HTML below does a few things:

- Provides a text area where users can input text to translate.
- Includes a language selector.
- Includes HTML elements to render the detected language and confidence scores returned during translation.
- Provides a read-only text area where the translation output is displayed.
- Includes placeholders for sentiment analysis and speech synthesis code that you'll add to this file later in the tutorial.

Let's update `index.html`.

1. Open `index.html` and locate these code comments:

```
<!-- HTML provided in the following sections goes here. -->
<!-- End -->
```

2. Replace the code comments with this HTML block:

```

<div class="row">
 <div class="col">
 <form>
 <!-- Enter text to translate. -->
 <div class="form-group">
 <label for="text-to-translate">Enter the text you'd like to translate:</label>
 <textarea class="form-control" id="text-to-translate" rows="5"></textarea>
 </div>
 <!-- Select output language. -->
 <div class="form-group">
 <label for="select-language">Translate to:</label>
 <select class="form-control" id="select-language">
 <option value="ar">Arabic</option>
 <option value="ca">Catalan</option>
 <option value="zh-Hans">Chinese (Simplified)</option>
 <option value="zh-Hant">Chinese (Traditional)</option>
 <option value="hr">Croatian</option>
 <option value="en">English</option>
 <option value="fr">French</option>
 <option value="de">German</option>
 <option value="el">Greek</option>
 <option value="he">Hebrew</option>
 <option value="hi">Hindi</option>
 <option value="it">Italian</option>
 <option value="ja">Japanese</option>
 <option value="ko">Korean</option>
 <option value="pt">Portuguese</option>
 <option value="ru">Russian</option>
 <option value="es">Spanish</option>
 <option value="th">Thai</option>
 <option value="tr">Turkish</option>
 <option value="vi">Vietnamese</option>
 </select>
 </div>
 <button type="submit" class="btn btn-primary mb-2" id="translate">Translate text</button>

 <div id="detected-language" style="display: none">
 Detected language:

 Detection confidence:

 </div>

 <!-- Start sentiment code-->
 <!-- End sentiment code -->
 </form>
 </div>
 <div class="col">
 <!-- Translated text returned by the Translate API is rendered here. -->
 <form>
 <div class="form-group" id="translator-text-response">
 <label for="translation-result">Translated text:</label>
 <textarea readonly class="form-control" id="translation-result" rows="5"></textarea>
 </div>
 <!-- Start voice font selection code -->
 <!-- End voice font selection code -->
 </form>
 <!-- Add Speech Synthesis button and audio element -->
 <!-- End Speech Synthesis button -->
 </div>
</div>

```

The next step is to write some Javascript. This is the bridge between your HTML and Flask route.

### Create `main.js`

The `main.js` file is the bridge between your HTML and Flask route. Your app will use a combination of jQuery, Ajax, and XMLHttpRequest to render content, and make `POST` requests to your Flask routes.

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the contents of the text area and the language selector are assigned to variables, and then passed along in the request to `translate-text`.

The code then iterates through the response, and updates the HTML with the translation, detected language, and confidence score.

1. From your IDE, create a file named `main.js` in the `static/scripts` directory.
2. Copy this code into `static/scripts/main.js`:

```
//Initiate jQuery on load.
$(function() {
 //Translate text with flask route
 $("#translate").on("click", function(e) {
 e.preventDefault();
 var translateVal = document.getElementById("text-to-translate").value;
 var languageVal = document.getElementById("select-language").value;
 var translateRequest = { 'text': translateVal, 'to': languageVal }

 if (translateVal !== "") {
 $.ajax({
 url: '/translate-text',
 method: 'POST',
 headers: {
 'Content-Type': 'application/json'
 },
 dataType: 'json',
 data: JSON.stringify(translateRequest),
 success: function(data) {
 for (var i = 0; i < data.length; i++) {
 document.getElementById("translation-result").textContent = data[i].translations[0].text;
 document.getElementById("detected-language-result").textContent =
 data[i].detectedLanguage.language;
 if (document.getElementById("detected-language-result").textContent !== ""){
 document.getElementById("detected-language").style.display = "block";
 }
 document.getElementById("confidence").textContent = data[i].detectedLanguage.score;
 }
 }
 });
 }
 });
 // In the following sections, you'll add code for sentiment analysis and
 // speech synthesis here.
})
```

### Test translation

Let's test translation in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. If it doesn't work, make sure that you've added your subscription key.

### TIP

If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

Press **CTRL + c** to kill the app, then head to the next section.

## Analyze sentiment

The [Text Analytics API](#) can be used to perform sentiment analysis, extract key phrases from text, or detect the source language. In this app, we're going to use sentiment analysis to determine if the provided text is positive, neutral, or negative. The API returns a numeric score between 0 and 1. Scores close to 1 indicate positive sentiment, and scores close to 0 indicate negative sentiment.

In this section, you're going to do a few things:

- Write some Python to call the Text Analytics API to perform sentiment analysis and return a response
- Create a Flask route to call your Python code
- Update the HTML with an area for sentiment scores, and a button to perform analysis
- Write Javascript that allows users to interact with your Flask app from the HTML

### Call the Text Analytics API

Let's write a function to call the Text Analytics API. This function will take four arguments: `input_text`, `input_language`, `output_text`, and `output_language`. This function is called whenever a user presses the run sentiment analysis button in your app. Data provided by the user from the text area and language selector, as well as the detected language and translation output are provided with each request. The response object includes sentiment scores for the source and translation. In the following sections, you're going to write some Javascript to parse the response and use it in your app. For now, let's focus on call the Text Analytics API.

1. Let's create a file called `sentiment.py` in the root of your working directory.
2. Next, add this code to `sentiment.py`.

```

import os, requests, uuid, json

Don't forget to replace with your Cog Services subscription key!
subscription_key = 'YOUR_TEXT_ANALYTICS_SUBSCRIPTION_KEY'

Our Flask route will supply four arguments: input_text, input_language,
output_text, output_language.
When the run sentiment analysis button is pressed in our Flask app,
the Ajax request will grab these values from our web app, and use them
in the request. See main.js for Ajax calls.

def get_sentiment(input_text, input_language, output_text, output_language):
 base_url = 'https://westus.api.cognitive.microsoft.com/text/analytics'
 path = '/v2.0/sentiment'
 constructed_url = base_url + path

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

 # You can pass more than one object in body.
 body = {
 'documents': [
 {
 'language': input_language,
 'id': '1',
 'text': input_text
 },
 {
 'language': output_language,
 'id': '2',
 'text': output_text
 }
]
 }
 response = requests.post(constructed_url, headers=headers, json=body)
 return response.json()

```

### 3. Add your Text Analytics subscription key and save.

#### Add a route to `app.py`

Let's create a route in your Flask app that calls `sentiment.py`. This route will be called each time a user presses the run sentiment analysis button in your app. Like the route for translation, this route is going to accept `POST` requests since the function expects arguments.

#### 1. Open `app.py` and locate the import statement at the top of `app.py` and update it:

```
import translate, sentiment
```

Now our Flask app can use the method available via `sentiment.py`.

#### 2. Copy this code to the end of `app.py` and save:

```
@app.route('/sentiment-analysis', methods=['POST'])
def sentiment_analysis():
 data = request.get_json()
 input_text = data['inputText']
 input_lang = data['inputLanguage']
 output_text = data['outputText']
 output_lang = data['outputLanguage']
 response = sentiment.get_sentiment(input_text, input_lang, output_text, output_lang)
 return jsonify(response)
```

## Update `index.html`

Now that you have a function to run sentiment analysis, and a route in your Flask app to call it, the next step is to start writing the HTML for your app. The HTML below does a few things:

- Adds a button to your app to run sentiment analysis
- Adds an element that explains sentiment scoring
- Adds an element to display the sentiment scores

1. Open `index.html` and locate these code comments:

```
<!-- Start sentiment code-->

<!-- End sentiment code -->
```

2. Replace the code comments with this HTML block:

```
<button type="submit" class="btn btn-primary mb-2" id="sentiment-analysis">Run sentiment
analysis</button>

<div id="sentiment" style="display: none">
 <p>Sentiment scores are provided on a 1 point scale. The closer the sentiment score is to 1,
 indicates positive sentiment. The closer it is to 0, indicates negative sentiment.</p>
 Sentiment score for input:

 Sentiment score for translation:
</div>
```

## Update `main.js`

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the contents of the text area and the language selector are assigned to variables, and then passed along in the request to the `sentiment-analysis` route.

The code then iterates through the response, and updates the HTML with the sentiment scores.

1. From your IDE, create a file named `main.js` in the `static` directory.
2. Copy this code into `static/scripts/main.js`:

```

//Run sentiment analysis on input and translation.
$("#sentiment-analysis").on("click", function(e) {
 e.preventDefault();
 var inputText = document.getElementById("text-to-translate").value;
 var inputLanguage = document.getElementById("detected-language-result").innerHTML;
 var outputText = document.getElementById("translation-result").value;
 var outputLanguage = document.getElementById("select-language").value;

 var sentimentRequest = { "inputText": inputText, "inputLanguage": inputLanguage, "outputText": outputText, "outputLanguage": outputLanguage };

 if (inputText !== "") {
 $.ajax({
 url: "/sentiment-analysis",
 method: "POST",
 headers: {
 "Content-Type": "application/json"
 },
 dataType: "json",
 data: JSON.stringify(sentimentRequest),
 success: function(data) {
 for (var i = 0; i < data.documents.length; i++) {
 if (typeof data.documents[i] !== "undefined"){
 if (data.documents[i].id === "1") {
 document.getElementById("input-sentiment").textContent = data.documents[i].score;
 }
 if (data.documents[i].id === "2") {
 document.getElementById("translation-sentiment").textContent = data.documents[i].score;
 }
 }
 }
 for (var i = 0; i < data.errors.length; i++) {
 if (typeof data.errors[i] !== "undefined"){
 if (data.errors[i].id === "1") {
 document.getElementById("input-sentiment").textContent = data.errors[i].message;
 }
 if (data.errors[i].id === "2") {
 document.getElementById("translation-sentiment").textContent = data.errors[i].message;
 }
 }
 }
 if (document.getElementById("input-sentiment").textContent !== '' &&
 document.getElementById("translation-sentiment").textContent !== ""){
 document.getElementById("sentiment").style.display = "block";
 }
 }
 });
 }
});
// In the next section, you'll add code for speech synthesis here.

```

## Test sentiment analysis

Let's test sentiment analysis in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. Next, press the run sentiment analysis button. You should see two scores. If it doesn't work, make sure that you've added your subscription key.

### TIP

If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

Press **CTRL + c** to kill the app, then head to the next section.

## Convert text-to-speech

The [Text-to-speech API](#) enables your app to convert text into natural human-like synthesized speech. The service supports standard, neural, and custom voices. Our sample app uses a handful of the available voices, for a full list, see [supported languages](#).

In this section, you're going to do a few things:

- Write some Python to convert text-to-speech with the Text-to-speech API
- Create a Flask route to call your Python code
- Update the HTML with a button to convert text-to-speech, and an element for audio playback
- Write Javascript that allows users to interact with your Flask app

### Call the Text-to-Speech API

Let's write a function to convert text-to-speech. This function will take two arguments: `input_text` and `voice_font`. This function is called whenever a user presses the convert text-to-speech button in your app. `input_text` is the translation output returned by the call to translate text, `voice_font` is the value from the voice font selector in the HTML.

1. Let's create a file called `synthesize.py` in the root of your working directory.
2. Next, add this code to `synthesize.py`.

```

import os, requests, time
from xml.etree import ElementTree

class TextToSpeech(object):
 def __init__(self, input_text, voice_font):
 subscription_key = 'YOUR_SPEECH_SERVICES_SUBSCRIPTION_KEY'
 self.subscription_key = subscription_key
 self.input_text = input_text
 self.voice_font = voice_font
 self.timestr = time.strftime('%Y%m%d-%H%M')
 self.access_token = None

 # This function performs the token exchange.
 def get_token(self):
 fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
 headers = {
 'Ocp-Apim-Subscription-Key': self.subscription_key
 }
 response = requests.post(fetch_token_url, headers=headers)
 self.access_token = str(response.text)

 # This function calls the TTS endpoint with the access token.
 def save_audio(self):
 base_url = 'https://westus.tts.speech.microsoft.com/'
 path = 'cognitiveservices/v1'
 constructed_url = base_url + path
 headers = {
 'Authorization': 'Bearer ' + self.access_token,
 'Content-Type': 'application/ssml+xml',
 'X-Microsoft-OutputFormat': 'riff-24kHz-16bit-mono-pcm',
 'User-Agent': 'YOUR_RESOURCE_NAME',
 }
 # Build the SSML request with ElementTree
 xml_body = ElementTree.Element('speak', version='1.0')
 xml_body.set('{http://www.w3.org/XML/1998/namespace}lang', 'en-us')
 voice = ElementTree.SubElement(xml_body, 'voice')
 voice.set('{http://www.w3.org/XML/1998/namespace}lang', 'en-US')
 voice.set('name', 'Microsoft Server Speech Text to Speech Voice {}'.format(self.voice_font))
 voice.text = self.input_text
 # The body must be encoded as UTF-8 to handle non-ascii characters.
 body = ElementTree.tostring(xml_body, encoding="utf-8")

 #Send the request
 response = requests.post(constructed_url, headers=headers, data=body)

 # Write the response as a wav file for playback. The file is located
 # in the same directory where this sample is run.
 return response.content

```

### 3. Add your Speech Services subscription key and save.

**Add a route to** `app.py`

Let's create a route in your Flask app that calls `synthesize.py`. This route will be called each time a user presses the convert text-to-speech button in your app. Like the routes for translation and sentiment analysis, this route is going to accept `POST` requests since the function expects two arguments: the text to synthesize, and the voice font for playback.

1. Open `app.py` and locate the import statement at the top of `app.py` and update it:

```
import translate, sentiment, synthesize
```

Now our Flask app can use the method available via `synthesize.py`.

2. Copy this code to the end of `app.py` and save:

```
@app.route('/text-to-speech', methods=['POST'])
def text_to_speech():
 data = request.get_json()
 text_input = data['text']
 voice_font = data['voice']
 tts = synthesize.TextToSpeech(text_input, voice_font)
 tts.get_token()
 audio_response = tts.save_audio()
 return audio_response
```

**Update** `index.html`

Now that you have a function to convert text-to-speech, and a route in your Flask app to call it, the next step is to start writing the HTML for your app. The HTML below does a few things:

- Provides a voice selection drop-down
- Adds a button to convert text-to-speech
- Adds an audio element, which is used to play back the synthesized speech

1. Open `index.html` and locate these code comments:

```
<!-- Start voice font selection code -->

<!-- End voice font selection code -->
```

2. Replace the code comments with this HTML block:

```

<div class="form-group">
 <label for="select-voice">Select voice font:</label>
 <select class="form-control" id="select-voice">
 <option value="(ar-SA, Naayf)">Arabic | Male | Naayf</option>
 <option value="(ca-ES, HerenaRUS)">Catalan | Female | HerenaRUS</option>
 <option value="(zh-CN, HuihuiRUS)">Chinese (Mainland) | Female | HuihuiRUS</option>
 <option value="(zh-CN, Kangkang, Apollo)">Chinese (Mainland) | Male | Kangkang, Apollo</option>
 <option value="(zh-HK, Tracy, Apollo)">Chinese (Hong Kong) | Female | Tracy, Apollo</option>
 <option value="(zh-HK, Danny, Apollo)">Chinese (Hong Kong) | Male | Danny, Apollo</option>
 <option value="(zh-TW, Yating, Apollo)">Chinese (Taiwan) | Female | Yaiting, Apollo</option>
 <option value="(zh-TW, Zhiwei, Apollo)">Chinese (Taiwan) | Male | Zhiwei, Apollo</option>
 <option value="(hr-HR, Matej)">Croatian | Male | Matej</option>
 <option value="(en-US, Jessa24kRUS)">English (US) | Female | Jessa24kRUS</option>
 <option value="(en-US, Guy24kRUS)">English (US) | Male | Guy24kRUS</option>
 <option value="(en-IE, Sean)">English (IE) | Male | Sean</option>
 <option value="(fr-FR, Julie, Apollo)">French | Female | Julie, Apollo</option>
 <option value="(fr-FR, HortenseRUS)">French | Female | Julie, HortenseRUS</option>
 <option value="(fr-FR, Paul, Apollo)">French | Male | Paul, Apollo</option>
 <option value="(de-DE, Hedda)">German | Female | Hedda</option>
 <option value="(de-DE, HeddaRUS)">German | Female | HeddaRUS</option>
 <option value="(de-DE, Stefan, Apollo)">German | Male | Apollo</option>
 <option value="(el-GR, Stefanos)">Greek | Male | Stefanos</option>
 <option value="(he-IL, Asaf)">Hebrew (Isreal) | Male | Asaf</option>
 <option value="(hi-IN, Kalpana, Apollo)">Hindi | Female | Kalpana, Apollo</option>
 <option value="(hi-IN, Hemant)">Hindi | Male | Hemant</option>
 <option value="(it-IT, LuciaRUS)">Italian | Female | LuciaRUS</option>
 <option value="(it-IT, Cosimo, Apollo)">Italian | Male | Cosimo, Apollo</option>
 <option value="(ja-JP, Ichiro, Apollo)">Japanese | Male | Ichiro</option>
 <option value="(ja-JP, HarukaRUS)">Japanese | Female | HarukaRUS</option>
 <option value="(ko-KR, HeamiRUS)">Korean | Female | Haemi</option>
 <option value="(pt-BR, HeloisaRUS)">Portuguese (Brazil) | Female | HeloisaRUS</option>
 <option value="(pt-BR, Daniel, Apollo)">Portuguese (Brazil) | Male | Daniel, Apollo</option>
 <option value="(pt-PT, HeliaRUS)">Portuguese (Portugal) | Female | HeliaRUS</option>
 <option value="(ru-RU, Irina, Apollo)">Russian | Female | Irina, Apollo</option>
 <option value="(ru-RU, Pavel, Apollo)">Russian | Male | Pavel, Apollo</option>
 <option value="(ru-RU, EkaterinaRUS)">Russian | Female | EkaterinaRUS</option>
 <option value="(es-ES, Laura, Apollo)">Spanish | Female | Laura, Apollo</option>
 <option value="(es-ES, HelenaRUS)">Spanish | Female | HelenaRUS</option>
 <option value="(es-ES, Pablo, Apollo)">Spanish | Male | Pablo, Apollo</option>
 <option value="(th-TH, Pattara)">Thai | Male | Pattara</option>
 <option value="(tr-TR, SedaRUS)">Turkish | Female | SedaRUS</option>
 <option value="(vi-VN, An)">Vietnamese | Male | An</option>
 </select>
</div>

```

3. Next, locate these code comments:

```

<!-- Add Speech Synthesis button and audio element -->

<!-- End Speech Synthesis button -->

```

4. Replace the code comments with this HTML block:

```

<button type="submit" class="btn btn-primary mb-2" id="text-to-speech">Convert text-to-speech</button>
<div id="audio-playback">
 <audio id="audio" controls>
 <source id="audio-source" type="audio/mpeg" />
 </audio>
</div>

```

5. Make sure to save your work.

**Update** main.js

In the code below, content from the HTML is used to construct a request to your Flask route. Specifically, the translation and the voice font are assigned to variables, and then passed along in the request to the `text-to-speech` route.

The code then iterates through the response, and updates the HTML with the sentiment scores.

1. From your IDE, create a file named `main.js` in the `static` directory.
2. Copy this code into `static/scripts/main.js` :

```
// Convert text-to-speech
$("#text-to-speech").on("click", function(e) {
 e.preventDefault();
 var ttsInput = document.getElementById("translation-result").value;
 var ttsVoice = document.getElementById("select-voice").value;
 var ttsRequest = { 'text': ttsInput, 'voice': ttsVoice }

 var xhr = new XMLHttpRequest();
 xhr.open("post", "/text-to-speech", true);
 xhr.setRequestHeader("Content-Type", "application/json");
 xhr.responseType = "blob";
 xhr.onload = function(evt){
 if (xhr.status === 200) {
 audioBlob = new Blob([xhr.response], {type: "audio/mpeg"});
 audioURL = URL.createObjectURL(audioBlob);
 if (audioURL.length > 5){
 var audio = document.getElementById("audio");
 var source = document.getElementById("audio-source");
 source.src = audioURL;
 audio.load();
 audio.play();
 }else{
 console.log("An error occurred getting and playing the audio.")
 }
 }
 }
 xhr.send(JSON.stringify(ttsRequest));
});
// Code for automatic language selection goes here.
```

3. You're almost done. The last thing you're going to do is add some code to `main.js` to automatically select a voice font based on the language selected for translation. Add this code block to `main.js` :

```

// Automatic voice font selection based on translation output.
$('select[id="select-language"]').change(function(e) {
 if ($(this).val() == "ar"){
 document.getElementById("select-voice").value = "(ar-SA, Naayf)";
 }
 if ($(this).val() == "ca"){
 document.getElementById("select-voice").value = "(ca-ES, HerenaRUS)";
 }
 if ($(this).val() == "zh-Hans"){
 document.getElementById("select-voice").value = "(zh-HK, Tracy, Apollo)";
 }
 if ($(this).val() == "zh-Hant"){
 document.getElementById("select-voice").value = "(zh-HK, Tracy, Apollo)";
 }
 if ($(this).val() == "hr"){
 document.getElementById("select-voice").value = "(hr-HR, Matej)";
 }
 if ($(this).val() == "en"){
 document.getElementById("select-voice").value = "(en-US, Jessa24kRUS)";
 }
 if ($(this).val() == "fr"){
 document.getElementById("select-voice").value = "(fr-FR, HortenseRUS)";
 }
 if ($(this).val() == "de"){
 document.getElementById("select-voice").value = "(de-DE, HeddaRUS)";
 }
 if ($(this).val() == "el"){
 document.getElementById("select-voice").value = "(el-GR, Stefanos)";
 }
 if ($(this).val() == "he"){
 document.getElementById("select-voice").value = "(he-IL, Asaf)";
 }
 if ($(this).val() == "hi"){
 document.getElementById("select-voice").value = "(hi-IN, Kalpana, Apollo)";
 }
 if ($(this).val() == "it"){
 document.getElementById("select-voice").value = "(it-IT, LuciaRUS)";
 }
 if ($(this).val() == "ja"){
 document.getElementById("select-voice").value = "(ja-JP, HarukaRUS)";
 }
 if ($(this).val() == "ko"){
 document.getElementById("select-voice").value = "(ko-KR, HeamiRUS)";
 }
 if ($(this).val() == "pt"){
 document.getElementById("select-voice").value = "(pt-BR, HeloisaRUS)";
 }
 if ($(this).val() == "ru"){
 document.getElementById("select-voice").value = "(ru-RU, EkaterinaRUS)";
 }
 if ($(this).val() == "es"){
 document.getElementById("select-voice").value = "(es-ES, HelenaRUS)";
 }
 if ($(this).val() == "th"){
 document.getElementById("select-voice").value = "(th-TH, Pattara)";
 }
 if ($(this).val() == "tr"){
 document.getElementById("select-voice").value = "(tr-TR, SedaRUS)";
 }
 if ($(this).val() == "vi"){
 document.getElementById("select-voice").value = "(vi-VN, An)";
 }
});

```

## Test your app

Let's test speech synthesis in the app.

```
flask run
```

Navigate to the provided server address. Type text into the input area, select a language, and press translate. You should get a translation. Next, select a voice, then press the convert text-to-speech button. the translation should be played back as synthesized speech. If it doesn't work, make sure that you've added your subscription key.

**TIP**

If the changes you've made aren't showing up, or the app doesn't work the way you expect it to, try clearing your cache or opening a private/incognito window.

That's it, you have a working app that performs translations, analyzes sentiment, and synthesized speech. Press **CTRL + c** to kill the app. Be sure to check out the other [Azure Cognitive Services](#).

## Get the source code

The source code for this project is available on [GitHub](#).

## Next steps

- [Translator Text API reference](#)
- [Text Analytics API reference](#)
- [Text-to-speech API reference](#)

# Release notes

12/16/2019 • 14 minutes to read • [Edit Online](#)

## Speech SDK 1.8.0: 2019-November release

### New Features

- Added a `FromHost()` API, to ease use with on-prem containers and sovereign clouds.
- Added Automatic Source Language Detection for Speech Recognition (in Java and C++)
- Added `SourceLanguageConfig` object for Speech Recognition, used to specify expected source languages (in Java and C++)
- Added `KeywordRecognizer` support on Windows (UWP), Android and iOS through the Nuget and Unity packages
- Added Remote Conversation Java API to do Conversation Transcription in asynchronous batches.

### Breaking changes

- Conversation Transcriber functionalities moved under namespace  
`Microsoft.CognitiveServices.Speech.Transcription`.
- Part of the Conversation Transcriber methods are moved to new `Conversation` class.
- Dropped support for 32-bit (ARMv7 and x86) iOS

### Bug fixes

- Fix for crash if local `KeywordRecognizer` is used without a valid speech service subscription key

### Samples

- Xamarin sample for `KeywordRecognizer`
- Unity sample for `KeywordRecognizer`
- C++ and Java samples for Automatic Source Language Detection.

## Speech SDK 1.7.0: 2019-September release

### New Features

- Added beta support for Xamarin on Universal Windows Platform (UWP), Android, and iOS
- Added iOS support for Unity
- Added `Compressed` input support for ALaw, Mulaw, FLAC on Android, iOS and Linux
- Added `SendMessageAsync` in `Connection` class for sending a message to service
- Added `SetMessageProperty` in `Connection` class for setting property of a message
- TTS added bindings for Java (Jre and Android), Python, Swift, and Objective-C
- TTS added playback support for macOS, iOS, and Android.
- Added "word boundary" information for TTS.

### Bug fixes

- Fixed IL2CPP build issue on Unity 2019 for Android
- Fixed issue with malformed headers in wav file input being processed incorrectly
- Fixed issue with UUIDs not being unique in some connection properties

- Fixed a few warnings about nullability specifiers in the Swift bindings (might require small code changes)
- Fixed a bug that caused websocket connections to be closed ungracefully under network load
- Fixed an issue on Android that sometimes results in duplicate impression IDs used by `DialogServiceConnector`
- Improvements to the stability of connections across multi-turn interactions and the reporting of failures (via `Canceled` events) when they occur with `DialogServiceConnector`
- `DialogServiceConnector` session starts will now properly provide events, including when calling `ListenOnceAsync()` during an active `StartKeywordRecognitionAsync()`
- Addressed a crash associated with `DialogServiceConnector` activities being received

## Samples

- Quickstart for Xamarin
- Updated CPP Quickstart with Linux ARM64 information
- Updated Unity quickstart with iOS information

# Speech SDK 1.6.0: 2019-June release

## Samples

- Quickstart samples for Text To Speech on UWP and Unity
- Quickstart sample for Swift on iOS
- Unity samples for Speech & Intent Recognition and Translation
- Updated quickstart samples for `DialogServiceConnector`

## Improvements / Changes

- Dialog namespace:
  - `SpeechBotConnector` has been renamed to `DialogServiceConnector`
  - `BotConfig` has been renamed to `DialogServiceConfig`
  - `BotConfig::FromChannelSecret()` has been remapped to `DialogServiceConfig::FromBotSecret()`
  - All existing Direct Line Speech clients continue to be supported after the rename
- Update TTS REST adapter to support proxy, persistent connection
- Improve error message when an invalid region is passed
- Swift/Objective-C:
  - Improved error reporting: Methods that can result in an error are now present in two versions: One that exposes an `NSError` object for error handling, and one that raises an exception. The former are exposed to Swift. This change requires adaptations to existing Swift code.
  - Improved event handling

## Bug fixes

- Fix for TTS: where `SpeakTextAsync` future returned without waiting until audio has completed rendering
- Fix for marshaling strings in C# to enable full language support
- Fix for .NET core app problem to load core library with net461 target framework in samples
- Fix for occasional issues to deploy native libraries to the output folder in samples
- Fix for web socket closing reliably
- Fix for possible crash while opening a connection under very heavy load on Linux
- Fix for missing metadata in the framework bundle for macOS
- Fix for problems with `pip install --user` on Windows

# Speech SDK 1.5.1

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

## Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

# Speech SDK 1.5.0: 2019-May release

## New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see [here](#).
- Conversation transcription functionality is available through the SDK. See [here](#).
- Add support for voice assistants using the Direct Line Speech channel.

## Samples

- Added samples for new features or new services supported by the SDK.

## Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `outputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

## Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

# Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

# Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

# Speech SDK 1.4.0: 2019-April release

## New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the [text-to-speech overview](#).
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details [here](#)).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C

support for macOS is currently in beta.

- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.
- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

## Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

## Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details [here](#)).
- JavaScript: Improve audio processing performance.

## Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

## Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

### Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

## Speech SDK 1.3.0: 2019-February release

### New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing [audio input device selection](#). This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the [GitHub sample repository](#). This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our [Unity quickstart](#).
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

## Samples

The following new content is available in our [sample repository](#):

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `connection` object in iOS.
- Additional Java samples for translation with audio output.

- New sample for use of the [Batch Transcription REST API](#).

## Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

## Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

## Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at `turn.end`, not at `speech.end`.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

## Speech SDK 1.2.0: 2018-December release

### New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see [here](#)](quickstart-python.md).
- JavaScript

- The Speech SDK for JavaScript has been open-sourced. The source code is available on [GitHub](#).
- We now support Node.js, more info can be found [here](#).
- The length restriction for audio sessions has been removed, reconnection will happen automatically under the cover.
- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

## Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

## Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; `RecognizedIntent` was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

## Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the [sample repository](#).

# Speech SDK 1.1.0

## New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

## Improvements

- Added additional verification in the recognizer configuration, and added additional error message.

- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

## Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in `AddIntent` in JavaScript, and added missing `AddIntent` JavaScript signature.

## Samples

- Added C++ and C# samples for pull and push stream usage in the [sample repository](#).

## Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our [sample repository](#), a new sample for JavaScript was added.

## Cognitive Services Speech SDK 1.0.0: 2018-September release

### New features

- Support for Objective-C on iOS. Check out our [Objective-C quickstart for iOS](#).
- Support for JavaScript in browser. Check out our [JavaScript quickstart](#).

### Breaking changes

- With this release, a number of breaking changes are introduced. Check [this page](#) for details.

## Cognitive Services Speech SDK 0.6.0: 2018-August release

### New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the [UWP quickstart](#).
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the [Java Runtime](#)

[Environment quickstart](#).

## Functional change

- Expose additional error detail information on connection errors.

## Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

## Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.
- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

# Cognitive Services Speech SDK 0.5.0: 2018-July release

## New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the [Android quickstart](#).
- Support .NET Standard 2.0 on Windows. Check out the [.NET Core quickstart](#).
- Experimental: Support UWP on Windows (version 1709 or later).
  - Check out the [UWP quickstart](#).
  - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

## Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support `AuthorizationToken` for creating factory instances.

## Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- `SpeechOutputFormat` in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

## Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a [regions](#) page to describe the supported regions.

## Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

## Cognitive Services Speech SDK 0.4.0: 2018-June release

### Functional changes

- `AudioInputStream`

A recognizer now can consume a stream as the audio source. For more information, see the related [how-to guide](#).

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

### Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

### Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- Fired a `NoMatch` result when the response from service is timed out.
- The media foundation libraries on Windows are delay loaded. This library is required for microphone input only.
- The upload speed for audio data is limited to about twice the original audio speed.
- On Windows, C# .NET assemblies now are strong named.
- Documentation fix: `Region` is required information to create a recognizer.

More samples have been added and are constantly being updated. For the latest set of samples, see the [Speech SDK samples GitHub repository](#).

## Cognitive Services Speech SDK 0.2.12733: 2018-May release

This release is the first public preview release of the Cognitive Services Speech SDK.

# What is Conversation Transcription (Preview)?

12/4/2019 • 3 minutes to read • [Edit Online](#)

Conversation Transcription is a [speech-to-text](#) solution that combines speech recognition, speaker identification, and sentence attribution to each speaker (also known as *diarization*) to provide real-time and/or asynchronous transcription of any conversation. Conversation Transcription distinguishes speakers in a conversation to determine who said what and when, and makes it easy for developers to add speech-to-text to their applications that perform multi-speaker diarization.

## Key features

- **Timestamps** – each speaker utterance has a timestamp, so that you can easily find when a phrase was said.
- **Readable transcripts** – transcripts have formatting and punctuation added automatically to ensure the text closely matches what was being said.
- **User profiles** – user profiles are generated by collecting user voice samples and sending them to signature generation.
- **Speaker identification** – speakers are identified using user profiles and a *speaker identifier* is assigned to each.
- **Multi-speaker diarization** – determine who said what by synthesizing the audio stream with each speaker identifier.
- **Real-time transcription** – provide live transcripts of who is saying what and when while the conversation is happening.
- **asynchronous transcription** – provide transcripts with higher accuracy by using a multichannel audio stream.

### NOTE

Although Conversation Transcription does not put a limit on the number of speakers in the room, it is optimized for 2-10 speakers per session.

## Use cases

### Inclusive meetings

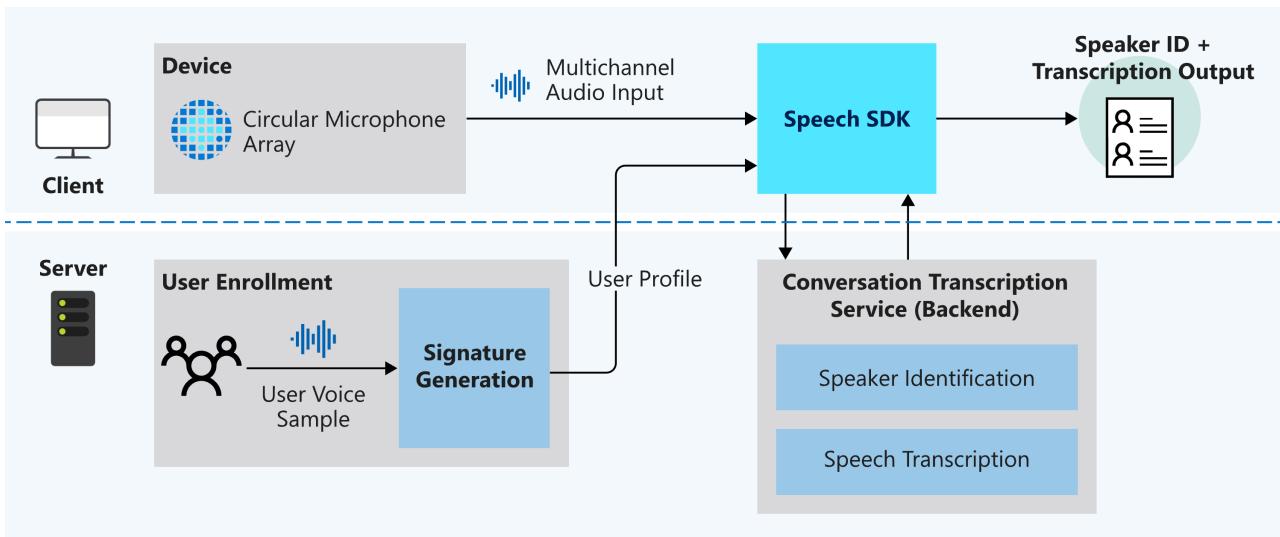
To make meetings inclusive for everyone, such as participants who are deaf and hard of hearing, it is important to have transcription in real time. Conversation Transcription in real-time mode takes meeting audio and determines who is saying what, allowing all meeting participants to follow the transcript and participate in the meeting without a delay.

### Improved efficiency

Meeting participants can focus on the meeting and leave note-taking to Conversation Transcription. Participants can actively engage in the meeting and quickly follow up on next steps, using the transcript instead of taking notes and potentially missing something during the meeting.

## How it works

This is a high-level overview of how Conversation Transcription works.



## Expected inputs

- **Multi-channel audio stream** – For specification and design details, see [Microsoft Speech Device SDK Microphone](#). To learn more or purchase a development kit, see [Get Microsoft Speech Device SDK](#).
- **User voice samples** – Conversation Transcription needs user profiles in advance of the conversation. You will need to collect audio recordings from each user, then send the recordings to the [Signature Generation Service](#) to validate the audio and generate user profiles.

## Real-time vs. asynchronous

Conversation Transcription offers three transcription modes:

### Real-time

Audio data is processed live to return speaker identifier + transcript. Select this mode if your transcription solution requirement is to provide conversation participants a live transcript view of their ongoing conversation. For example, building an application to make meetings more accessible the deaf and hard of hearing participants is an ideal use case for real-time transcription.

### Asynchronous

Audio data is batch processed to return speaker identifier and transcript. Select this mode if your transcription solution requirement is to provide higher accuracy without live transcript view. For example, if you want to build an application to allow meeting participants to easily catch up on missed meetings, then use the asynchronous transcription mode to get high-accuracy transcription results.

### Real-time plus asynchronous

Audio data is processed live to return speaker identifier + transcript, and, in addition, a request is created to also get a high-accuracy transcript through asynchronous processing. Select this mode if your application has a need for real-time transcription but also requires a higher accuracy transcript for use after the conversation or meeting occurred.

## Language support

Currently, Conversation Transcription supports "en-US" and "zh-CN" in the following regions:*centralus* and *eastasia*. If you require additional locale support, contact the [Conversation Transcription Feature Crew](#).

## Next steps

[Transcribe conversations in real time](#)

# Real time Conversation Transcription (Preview)

12/4/2019 • 4 minutes to read • [Edit Online](#)

The Speech SDK's **ConversationTranscriber** API allows you to transcribe meetings and other conversations with the ability to add, remove, and identify multiple participants by streaming audio to the Speech service using `PullStream` or `PushStream`. This topic requires you to know how to use Speech-to-text with the Speech SDK (version 1.8.0 or later). For more information, see [What are Speech services](#).

## Limitations

- The ConversationTranscriber API is supported for C++, C#, and Java on Windows, Linux, and Android.
- Currently available in "en-US" and "zh-CN" languages in the following regions: *centralus* and *eastasia*.
- Requires a 7-mic circular multi-microphone array with a playback reference stream. The microphone array should meet [our specification](#).
- The [Speech Devices SDK](#) provides suitable devices and a sample app demonstrating Conversation Transcription.

## Optional sample code resources

The Speech Device SDK provides sample code in Java for real-time audio capture using 8 channels.

- [ROOBO device sample code](#)
- [Azure Kinect Dev Kit sample code](#)

## Prerequisites

A Speech service subscription. You can [get a Speech trial subscription](#) if you don't have one.

## Create voice signatures

The first step is to create voice signatures for the conversation participants for efficient speaker identification.

### Audio input requirements

- The input audio wave file for creating voice signatures should be in 16-bit samples, 16 kHz sample rate, and a single channel (mono) format.
- The recommended length for each audio sample is between thirty seconds and two minutes.

### Sample code

The following example shows two different ways to create voice signature by [using the REST API](#) in C#. Note that you'll need to substitute real information for "YourSubscriptionKey", your wave file name for "speakerVoice.wav", and your region for `{region}` and "YourServiceRegion" (*centralus* or *eastasia*).

```

class Program
{
 static async Task CreateVoiceSignatureByUsingFormData()
 {
 // Replace with your own region
 var region = "YourServiceRegion";
 // Change the name of the wave file to match yours
 byte[] fileBytes = File.ReadAllBytes(@"speakerVoice.wav");
 var form = new MultipartFormDataContent();
 var content = new ByteArrayContent(fileBytes);
 form.Add(content, "file", "file");
 var client = new HttpClient();
 // Add your subscription key to the header Ocp-Apim-Subscription-Key directly
 // Replace "YourSubscriptionKey" with your own subscription key
 client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "YourSubscriptionKey");
 // Edit with your desired region for `'{region}'`
 var response = await client.PostAsync($"https://signature.{region}.cts.speech.microsoft.com/api/v1/Signature/GenerateVoiceSignatureFromFormData", form);
 // A voice signature contains Version, Tag and Data key values from the Signature json structure from the Response body.
 // Voice signature format example: { "Version": <Numeric value>, "Tag": "string", "Data": "string" }
 var jsonData = await response.Content.ReadAsStringAsync();
 }

 static async Task CreateVoiceSignatureByUsingBody()
 {
 // Replace with your own region
 var region = "YourServiceRegion";
 // Change the name of the wave file to match yours
 byte[] fileBytes = File.ReadAllBytes(@"speakerVoice.wav");
 var content = new ByteArrayContent(fileBytes);

 var client = new HttpClient();
 // Add your subscription key to the header Ocp-Apim-Subscription-Key directly
 // Replace "YourSubscriptionKey" with your own subscription key
 client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", "YourSubscriptionKey");
 // Edit with your desired region for `'{region}'`
 var response = await client.PostAsync($"https://signature.{region}.cts.speech.microsoft.com/api/v1/Signature/GenerateVoiceSignatureFromByteArray", content);
 // A voice signature contains Version, Tag and Data key values from the Signature json structure from the Response body.
 // Voice signature format example: { "Version": <Numeric value>, "Tag": "string", "Data": "string" }
 var jsonData = await response.Content.ReadAsStringAsync();
 }

 static void Main(string[] args)
 {
 CreateVoiceSignatureByUsingFormData().Wait();
 CreateVoiceSignatureByUsingBody().Wait();
 }
}

```

## Transcribe conversations

The following sample code demonstrates how to transcribe conversations in real time for three speakers. It assumes you've already created voice signatures for each speaker as shown above. Substitute real information for "YourSubscriptionKey" and "YourServiceRegion" when creating the SpeechConfig object.

Sample code highlights include:

- Creating a `Conversation` object from the `SpeechConfig` object using a meeting identifier generated using `Guid.NewGuid()`
- Creating a `ConversationTranscriber` object and join the conversation with `JoinConversationAsync()` to start

transcription

- Registering the events of interest
- Adding or removing participants to the conversation using the Conversation object
- Streaming the audio

The transcription and speaker identifier come back in the registered events.

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Transcription;

public class MyConversationTranscriber
{
 public static async Task ConversationWithPullAudioStreamAsync()
 {
 // Creates an instance of a speech config with specified subscription key and service region
 // Replace with your own subscription key and region
 var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
 config.SetProperty("ConversationTranscriptionInRoomAndOnline", "true");

 var stopTranscription = new TaskCompletionSource<int>();

 // Create an audio stream from a wav file or from the default microphone if you want to stream live
 // audio from the supported devices
 // Replace with your own audio file name and Helper class which implements AudioConfig using
 PullAudioInputStreamCallback
 using (var audioInput = Helper.OpenWavFile(@"8channelsOfRecordedPCMAudio.wav"))
 {
 var meetingId = Guid.NewGuid().ToString();
 using (var conversation = new Conversation(config, meetingId))
 {
 // Create a conversation transcriber using audio stream input
 using (var conversationTranscriber = new ConversationTranscriber (audioInput))
 {
 await conversationTranscriber.JoinConversationAsync(conversation);

 // Subscribe to events
 conversationTranscriber.Transcribing += (s, e) =>
 {
 Console.WriteLine($"TRANSCRIBING: Text={e.Result.Text}");
 };

 conversationTranscriber.Transcribed += (s, e) =>
 {
 if (e.Result.Reason == ResultReason.RecognizedSpeech)
 {
 Console.WriteLine($"TRANSCRIBED: Text={e.Result.Text}, UserID={e.Result.UserId}");
 }
 else if (e.Result.Reason == ResultReason.NoMatch)
 {
 Console.WriteLine($"NOMATCH: Speech could not be recognized.");
 }
 };
 };

 conversationTranscriber.Canceled += (s, e) =>
 {
 Console.WriteLine($"CANCELED: Reason={e.Reason}");

 if (e.Reason == CancellationReason.Error)
 {
 Console.WriteLine($"CANCELED: ErrorCode={e.ErrorCode}");
 Console.WriteLine($"CANCELED: ErrorDetails={e.ErrorDetails}");
 Console.WriteLine($"CANCELED: Did you update the subscription info?");

 stopTranscription.TrySetResult(0);
 }
 };
 };
 };
 };
}
```

```
 conversationTranscriber.SessionStarted += (s, e) =>
 {
 Console.WriteLine("\nSession started event.");
 };

 conversationTranscriber.SessionStopped += (s, e) =>
 {
 Console.WriteLine("\nSession stopped event.");
 Console.WriteLine("\nStop recognition.");
 stopTranscription.TrySetResult(0);
 };

 // Add participants to the conversation.
 // Create voice signatures using REST API described in the earlier section in this
document.

 // Voice signature needs to be in the following format:
 // { "Version": <Numeric value>, "Tag": "string", "Data": "string" }

 var speakerA = Participant.From("Speaker_A", "en-us", signatureA);
 var speakerB = Participant.From("Speaker_B", "en-us", signatureB);
 var speakerC = Participant.From("Speaker_C", "en-us", signatureC);
 await conversation.AddParticipantAsync(speakerA);
 await conversation.AddParticipantAsync(speakerB);
 await conversation.AddParticipantAsync(speakerC);

 // Starts transcribing of the conversation. Uses StopTranscribingAsync() to stop
transcribing when all participants leave.
 await conversationTranscriber.StartTranscribingAsync().ConfigureAwait(false);

 // Waits for completion.
 // Use Task.WaitAny to keep the task rooted.
 Task.WaitAny(new[] { stopTranscription.Task });

 // Stop transcribing the conversation.
 await conversationTranscriber.StopTranscribingAsync().ConfigureAwait(false);
}

}
}
```

## Next steps

## Asynchronous Conversation Transcription

# Asynchronous Conversation Transcription (Preview)

12/4/2019 • 4 minutes to read • [Edit Online](#)

In this article, asynchronous Conversation Transcription is demonstrated using the **RemoteConversationTranscriptionClient** API. If you have configured Conversation Transcription to do asynchronous transcription and have a `conversationId`, you can obtain the transcription associated with that `conversationId` using the **RemoteConversationTranscriptionClient** API.

## Asynchronous vs. real-time + asynchronous

With asynchronous transcription, you stream the conversation audio, but don't need a transcription returned in real time. Instead, after the audio is sent, use the `conversationId` of `Conversation` to query for the status of the asynchronous transcription. When the asynchronous transcription is ready, you'll get a `RemoteConversationTranscriptionResult`.

With real-time plus asynchronous, you get the transcription in real time, but also get the transcription by querying with the `conversationId` (similar to asynchronous scenario).

Two steps are required to accomplish asynchronous transcription. The first step is to upload the audio, choosing either asynchronous only or real-time plus asynchronous. The second step is to get the transcription results.

## Upload the audio

Before asynchronous transcription can be performed, you need to send the audio to Conversation Transcription Service using Microsoft Cognitive Speech client SDK (version 1.8.0 or above).

This example code shows how to create conversation transcriber for asynchronous-only mode. In order to stream audio to the transcriber, you will need to add audio streaming code derived from [Transcribe conversations in real time with the Speech SDK](#). Refer to the **Limitations** section of that topic to see the supported platforms and languages APIs.

```
// Create the speech config object
// Substitute real information for "YourSubscriptionKey" and "Region"
SpeechConfig speechConfig = SpeechConfig.fromSubscription("YourSubscriptionKey", "Region");
speechConfig.setProperty("ConversationTranscriptionInRoomAndOnline", "true");

// Set the property for asynchronous transcription
speechConfig.setServiceProperty("transcriptionMode", "Async", ServicePropertyChannel.UriQueryParameter);

// Set the property for real-time plus asynchronous transcription
//speechConfig.setServiceProperty("transcriptionMode", "RealTimeAndAsync",
//ServicePropertyChannel.UriQueryParameter);

// pick a conversation Id that is a GUID.
String conversationId = UUID.randomUUID().toString();

// Create a Conversation
Conversation conversation = new Conversation(speechConfig, conversationId);

// Create an audio stream from a wav file or from the default microphone if you want to stream live audio from
// the supported devices
// Replace with your own audio file name and Helper class which implements AudioConfig using
PullAudioInputStreamCallback
PullAudioInputStreamCallback wavfilePullStreamCallback =
Helper.OpenWavFile("16Khz16Bits8channelsOfRecordedPCMAudio.wav");
// Create an audio stream format assuming the file used above is 16Khz, 16 bits and 8 channel pcm wav file
```

```

AudioStreamFormat audioStreamFormat = AudioStreamFormat.getWaveFormatPCM((long)16000, (short)16,(short)8);
// Create an input stream
AudioInputStream audioStream = AudioInputStream.createPullStream(wavfilePullStreamCallback,
audioStreamFormat);

// Create a conversation transcriber
ConversationTranscriber transcriber = new ConversationTranscriber(AudioConfig.fromStreamInput(audioStream));

// join a conversation
transcriber.joinConversationAsync(conversation);

// Add the event listener for the realtime events
transcriber.transcribed.addEventListerner((o, e) -> {
 System.out.println("Conversation transcriber Recognized:" + e.toString());
});

transcriber.canceled.addEventListerner((o, e) -> {
 System.out.println("Conversation transcriber canceled:" + e.toString());
 try {
 transcriber.stopTranscribingAsync().get();
 } catch (InterruptedException ex) {
 ex.printStackTrace();
 } catch (ExecutionException ex) {
 ex.printStackTrace();
 }
});

transcriber.sessionStopped.addEventListerner((o, e) -> {
 System.out.println("Conversation transcriber stopped:" + e.toString());

 try {
 transcriber.stopTranscribingAsync().get();
 } catch (InterruptedException ex) {
 ex.printStackTrace();
 } catch (ExecutionException ex) {
 ex.printStackTrace();
 }
});

// start the transcription.
Future<?> future = transcriber.startTranscribingAsync();
...

```

If you want real-time *plus* asynchronous, comment and uncomment the appropriate lines of code as follows:

```

// Set the property for asynchronous transcription
//speechConfig.setServiceProperty("transcriptionMode", "Async", ServicePropertyChannel.UriQueryParameter);

// Set the property for real-time plus asynchronous transcription
speechConfig.setServiceProperty("transcriptionMode", "RealTimeAndAsync",
ServicePropertyChannel.UriQueryParameter);

```

## Get transcription results

This step gets the asynchronous transcription results but assumes any real-time processing you might have required is done elsewhere. For more information, see [Transcribe conversations in real time with the Speech SDK](#).

For the code shown here, you need **remote-conversation version 1.8.0**, supported only for Java (1.8.0 or above) on Windows, Linux, and Android (API level 26 or above only).

### Obtaining the client SDK

You can obtain **remote-conversation** by editing your pom.xml file as follows.

- At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK:

```
<repositories>
 <repository>
 <id>maven-cognitiveservices-speech</id>
 <name>Microsoft Cognitive Services Speech Maven Repository</name>
 <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
 </repository>
</repositories>
```

- Also add a `dependencies` element, with the `remoteconversation-client-sdk` 1.8.0 as a dependency:

```
<dependencies>
 <dependency>
 <groupId>com.microsoft.cognitiveservices.speech.remoteconversation</groupId>
 <artifactId>remote-conversation</artifactId>
 <version>1.8.0</version>
 </dependency>
</dependencies>
```

- Save the changes

### Sample transcription code

After you have the `conversationId`, create a remote conversation transcription client

**RemoteConversationTranscriptionClient** at the client application to query the status of the asynchronous transcription. Use `getTranscriptionOperation` method in **RemoteConversationTranscriptionClient** to get a **PollerFlux** object. The PollerFlux object will have information about the remote operation status **RemoteConversationTranscriptionOperation** and the final result **RemoteConversationTranscriptionResult**. Once the operation has finished, get **RemoteConversationTranscriptionResult** by calling `getFinalResult` on a **SyncPoller**. In this code we simply print the result contents to system output.

```

// Create the speech config object
SpeechConfig speechConfig = SpeechConfig.fromSubscription("YourSubscriptionKey", "Region");

// Create a remote Conversation Transcription client
RemoteConversationTranscriptionClient client = new RemoteConversationTranscriptionClient(speechConfig);

// Get the PollerFlux for the remote operation
PollerFlux<RemoteConversationTranscriptionOperation, RemoteConversationTranscriptionResult>
remoteTranscriptionOperation = client.getTranscriptionOperation(conversationId);

// Subscribe to PollerFlux to get the remote operation status
remoteTranscriptionOperation.subscribe(
 pollResponse -> {
 System.out.println("Poll response status : " + pollResponse.getStatus());
 System.out.println("Poll response status : " + pollResponse.getValue().getServiceStatus());
 }
);

// Obtain the blocking operation using getSyncPoller
SyncPoller<RemoteConversationTranscriptionOperation, RemoteConversationTranscriptionResult> blockingOperation
= remoteTranscriptionOperation.getSyncPoller();

// Wait for the operation to finish
blockingOperation.waitForCompletion();

// Get the final result response
RemoteConversationTranscriptionResult resultResponse = blockingOperation.getFinalResult();

// Print the result
if(resultResponse != null) {
 if(resultResponse.getConversationTranscriptionResults() != null) {
 for (int i = 0; i < resultResponse.getConversationTranscriptionResults().size(); i++) {
 ConversationTranscriptionResult result =
resultResponse.getConversationTranscriptionResults().get(i);

System.out.println(result.getProperties().getProperty(PropertyId.SpeechServiceResponse_JsonResult.name()));

System.out.println(result.getProperties().getProperty(PropertyId.SpeechServiceResponse_JsonResult));
 System.out.println(result.getOffset());
 System.out.println(result.getDuration());
 System.out.println(result.getUserId());
 System.out.println(result.getReason());
 System.out.println(result.getResultId());
 System.out.println(result.getText());
 System.out.println(result.toString());
 }
}
}

System.out.println("Operation finished");

```

## Next steps

[Explore our samples on GitHub](#)

# Release notes

12/16/2019 • 14 minutes to read • [Edit Online](#)

## Speech SDK 1.8.0: 2019-November release

### New Features

- Added a `FromHost()` API, to ease use with on-prem containers and sovereign clouds.
- Added Automatic Source Language Detection for Speech Recognition (in Java and C++)
- Added `SourceLanguageConfig` object for Speech Recognition, used to specify expected source languages (in Java and C++)
- Added `KeywordRecognizer` support on Windows (UWP), Android and iOS through the Nuget and Unity packages
- Added Remote Conversation Java API to do Conversation Transcription in asynchronous batches.

### Breaking changes

- Conversation Transcriber functionalities moved under namespace  
`Microsoft.CognitiveServices.Speech.Transcription`.
- Part of the Conversation Transcriber methods are moved to new `Conversation` class.
- Dropped support for 32-bit (ARMv7 and x86) iOS

### Bug fixes

- Fix for crash if local `KeywordRecognizer` is used without a valid speech service subscription key

### Samples

- Xamarin sample for `KeywordRecognizer`
- Unity sample for `KeywordRecognizer`
- C++ and Java samples for Automatic Source Language Detection.

## Speech SDK 1.7.0: 2019-September release

### New Features

- Added beta support for Xamarin on Universal Windows Platform (UWP), Android, and iOS
- Added iOS support for Unity
- Added `Compressed` input support for ALaw, Mulaw, FLAC on Android, iOS and Linux
- Added `SendMessageAsync` in `Connection` class for sending a message to service
- Added `SetMessageProperty` in `Connection` class for setting property of a message
- TTS added bindings for Java (Jre and Android), Python, Swift, and Objective-C
- TTS added playback support for macOS, iOS, and Android.
- Added "word boundary" information for TTS.

### Bug fixes

- Fixed IL2CPP build issue on Unity 2019 for Android
- Fixed issue with malformed headers in wav file input being processed incorrectly
- Fixed issue with UUIDs not being unique in some connection properties

- Fixed a few warnings about nullability specifiers in the Swift bindings (might require small code changes)
- Fixed a bug that caused websocket connections to be closed ungracefully under network load
- Fixed an issue on Android that sometimes results in duplicate impression IDs used by `DialogServiceConnector`
- Improvements to the stability of connections across multi-turn interactions and the reporting of failures (via `Canceled` events) when they occur with `DialogServiceConnector`
- `DialogServiceConnector` session starts will now properly provide events, including when calling `ListenOnceAsync()` during an active `StartKeywordRecognitionAsync()`
- Addressed a crash associated with `DialogServiceConnector` activities being received

## Samples

- Quickstart for Xamarin
- Updated CPP Quickstart with Linux ARM64 information
- Updated Unity quickstart with iOS information

# Speech SDK 1.6.0: 2019-June release

## Samples

- Quickstart samples for Text To Speech on UWP and Unity
- Quickstart sample for Swift on iOS
- Unity samples for Speech & Intent Recognition and Translation
- Updated quickstart samples for `DialogServiceConnector`

## Improvements / Changes

- Dialog namespace:
  - `SpeechBotConnector` has been renamed to `DialogServiceConnector`
  - `BotConfig` has been renamed to `DialogServiceConfig`
  - `BotConfig::FromChannelSecret()` has been remapped to `DialogServiceConfig::FromBotSecret()`
  - All existing Direct Line Speech clients continue to be supported after the rename
- Update TTS REST adapter to support proxy, persistent connection
- Improve error message when an invalid region is passed
- Swift/Objective-C:
  - Improved error reporting: Methods that can result in an error are now present in two versions: One that exposes an `NSError` object for error handling, and one that raises an exception. The former are exposed to Swift. This change requires adaptations to existing Swift code.
  - Improved event handling

## Bug fixes

- Fix for TTS: where `SpeakTextAsync` future returned without waiting until audio has completed rendering
- Fix for marshaling strings in C# to enable full language support
- Fix for .NET core app problem to load core library with net461 target framework in samples
- Fix for occasional issues to deploy native libraries to the output folder in samples
- Fix for web socket closing reliably
- Fix for possible crash while opening a connection under very heavy load on Linux
- Fix for missing metadata in the framework bundle for macOS
- Fix for problems with `pip install --user` on Windows

# Speech SDK 1.5.1

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

## Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

# Speech SDK 1.5.0: 2019-May release

## New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see [here](#).
- Conversation transcription functionality is available through the SDK. See [here](#).
- Add support for voice assistants using the Direct Line Speech channel.

## Samples

- Added samples for new features or new services supported by the SDK.

## Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `outputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

## Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

# Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

# Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

# Speech SDK 1.4.0: 2019-April release

## New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the [text-to-speech overview](#).
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only on Linux from C++ and C# and is currently in beta (more details [here](#)).
- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C

support for macOS is currently in beta.

- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.
- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

## Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

## Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details [here](#)).
- JavaScript: Improve audio processing performance.

## Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

## Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

### Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

## Speech SDK 1.3.0: 2019-February release

### New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing [audio input device selection](#). This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the [GitHub sample repository](#). This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our [Unity quickstart](#).
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

## Samples

The following new content is available in our [sample repository](#):

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.
- Additional samples for using the `connection` object in iOS.
- Additional Java samples for translation with audio output.

- New sample for use of the [Batch Transcription REST API](#).

## Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

## Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

## Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at `turn.end`, not at `speech.end`.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

## Speech SDK 1.2.0: 2018-December release

### New Features

- Python
  - The Beta version of Python support (3.5 and above) is available with this release. For more information, see [here](#)](quickstart-python.md).
- JavaScript

- The Speech SDK for JavaScript has been open-sourced. The source code is available on [GitHub](#).
- We now support Node.js, more info can be found [here](#).
- The length restriction for audio sessions has been removed, reconnection will happen automatically under the cover.
- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

## Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

## Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; `RecognizedIntent` was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

## Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the [sample repository](#).

# Speech SDK 1.1.0

## New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information with the reported error, it will now be available in the new members.

## Improvements

- Added additional verification in the recognizer configuration, and added additional error message.

- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

## Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in `AddIntent` in JavaScript, and added missing `AddIntent` JavaScript signature.

## Samples

- Added C++ and C# samples for pull and push stream usage in the [sample repository](#).

## Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our [sample repository](#), a new sample for JavaScript was added.

## Cognitive Services Speech SDK 1.0.0: 2018-September release

### New features

- Support for Objective-C on iOS. Check out our [Objective-C quickstart for iOS](#).
- Support for JavaScript in browser. Check out our [JavaScript quickstart](#).

### Breaking changes

- With this release, a number of breaking changes are introduced. Check [this page](#) for details.

## Cognitive Services Speech SDK 0.6.0: 2018-August release

### New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the [UWP quickstart](#).
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the [Java Runtime](#)

[Environment quickstart](#).

## Functional change

- Expose additional error detail information on connection errors.

## Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

## Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.
- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

# Cognitive Services Speech SDK 0.5.0: 2018-July release

## New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the [Android quickstart](#).
- Support .NET Standard 2.0 on Windows. Check out the [.NET Core quickstart](#).
- Experimental: Support UWP on Windows (version 1709 or later).
  - Check out the [UWP quickstart](#).
  - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

## Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support `AuthorizationToken` for creating factory instances.

## Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- `SpeechOutputFormat` in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

## Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a [regions](#) page to describe the supported regions.

## Known issue

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

## Cognitive Services Speech SDK 0.4.0: 2018-June release

### Functional changes

- `AudioInputStream`

A recognizer now can consume a stream as the audio source. For more information, see the related [how-to guide](#).

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

### Breaking change

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

### Bug fixes

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- Fired a `NoMatch` result when the response from service is timed out.
- The media foundation libraries on Windows are delay loaded. This library is required for microphone input only.
- The upload speed for audio data is limited to about twice the original audio speed.
- On Windows, C# .NET assemblies now are strong named.
- Documentation fix: `Region` is required information to create a recognizer.

More samples have been added and are constantly being updated. For the latest set of samples, see the [Speech SDK samples GitHub repository](#).

## Cognitive Services Speech SDK 0.2.12733: 2018-May release

This release is the first public preview release of the Cognitive Services Speech SDK.

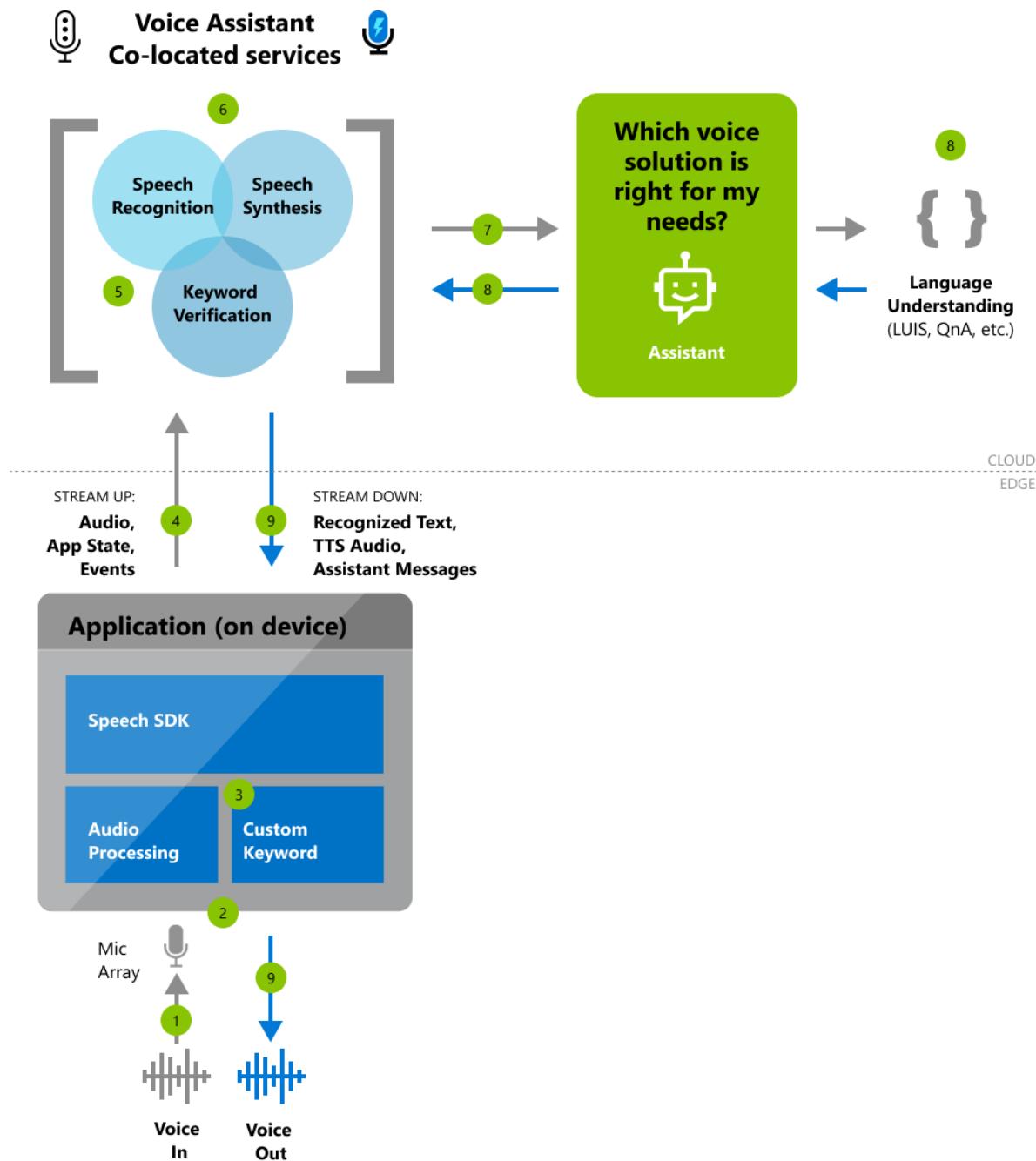
# About voice assistants

12/4/2019 • 3 minutes to read • [Edit Online](#)

Voice assistants using the Speech service empowers developers to create natural, human-like conversational interfaces for their applications and experiences.

The voice assistant service provides fast, reliable interaction between a device and an assistant implementation that uses either (1) the Bot Framework's Direct Line Speech channel or (2) the integrated Custom Commands (Preview) service for task completion.

Applications connect to the voice assistant service with the Speech Software Development Kit (SDK).



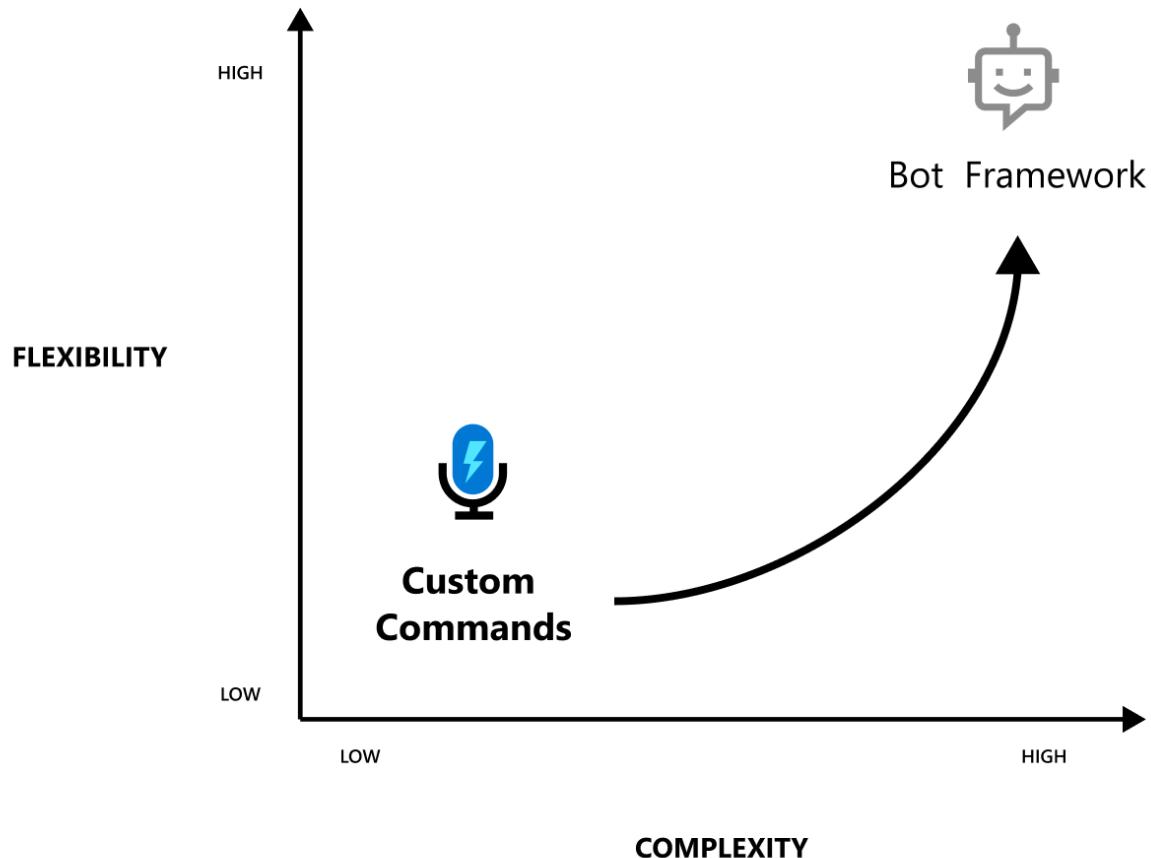
## Choosing an assistant solution

The first step to creating a voice assistant is to decide what it should do. The Speech service provides multiple, complementary solutions for crafting your assistant interactions. Whether you want the flexibility and versatility that the Bot Framework's [Direct Line Speech](#) channel provides or the simplicity of [Custom Commands \(Preview\)](#) for straightforward scenarios, selecting the right tools will get you started.

IF YOU WANT...	THEN CONSIDER...	FOR EXAMPLE...
Open-ended conversation with robust skills integration and full deployment control	The Bot Framework's <a href="#">Direct Line Speech</a> channel	<ul style="list-style-type: none"> <li>• "I need to go to Seattle"</li> <li>• "What kind of pizza can I order?"</li> </ul>
Command and control or task-oriented conversation with simplified authoring and hosting	<a href="#">Custom Commands (Preview)</a>	<ul style="list-style-type: none"> <li>• "Turn on the overhead light"</li> <li>• "Make it 5 degrees warmer"</li> </ul>

We recommend [Direct Line Speech](#) as the best default choice if you aren't yet sure what you'd like your assistant to handle. It offers integration with a rich set of tools and authoring aids such as the [Virtual Assistant Solution and Enterprise Template](#) and the [QnA Maker service](#) to build on common patterns and use your existing knowledge sources.

[Custom Commands \(Preview\)](#) provides a streamlined authoring and hosting experience specifically tailored for natural language command and control scenarios.



## Core features

Whether you choose [Direct Line Speech](#) or [Custom Commands \(Preview\)](#) to create your assistant interactions, you can use a rich set of customization features to customize your assistant to your brand, product, and personality.

CATEGORY	FEATURES
Custom keyword	Users can start conversations with assistants with a custom keyword like "Hey Contoso." An app does this with a custom keyword engine in the Speech SDK, which can be configured with a custom keyword <a href="#">that you can generate here</a> . Voice assistants can use service-side keyword verification to improve the accuracy of the keyword activation (versus the device alone).
Speech to text	Voice assistants convert real-time audio into recognized text using <a href="#">Speech-to-text</a> from the Speech service. This text is available, as it's transcribed, to both your assistant implementation and your client application.
Text to speech	Textual responses from your assistant are synthesized using <a href="#">Text-to-speech</a> from the Speech service. This synthesis is then made available to your client application as an audio stream. Microsoft offers the ability to build your own custom, high-quality Neural TTS voice that gives a voice to your brand. To learn more, <a href="#">contact us</a> .

## Getting started with voice assistants

We offer quickstarts designed to have you running code in less than 10 minutes. This table includes a list of voice assistant quickstarts, organized by language.

QUICKSTART	PLATFORM	API REFERENCE
C#, UWP	Windows	<a href="#">Browse</a>
Java	Windows, macOS, Linux	<a href="#">Browse</a>
Java	Android	<a href="#">Browse</a>

## Sample code

Sample code for creating a voice assistant is available on GitHub. These samples cover the client application for connecting to your assistant in several popular programming languages.

- [Voice assistant samples \(SDK\)](#)
- [Tutorial: Voice enable your assistant with the Speech SDK, C#](#)

## Tutorial

A tutorial on how to [voice-enable your assistant using the Speech SDK and Direct Line Speech channel](#).

## Customization

Voice assistants built using the Speech service can use the full range of customization options available for [speech-to-text](#), [text-to-speech](#), and [custom keyword selection](#).

**NOTE**

Customization options vary by language/locale (see [Supported languages](#)).

## Reference docs

- [Speech SDK](#)
- [Azure Bot Service](#)

## Next steps

- [Get a Speech service subscription key for free](#)
- [Get the Speech SDK](#)
- [Learn more about Custom Commands \(Preview\)](#)
- [Learn more about Direct Line Speech](#)

# Custom Commands (Preview)

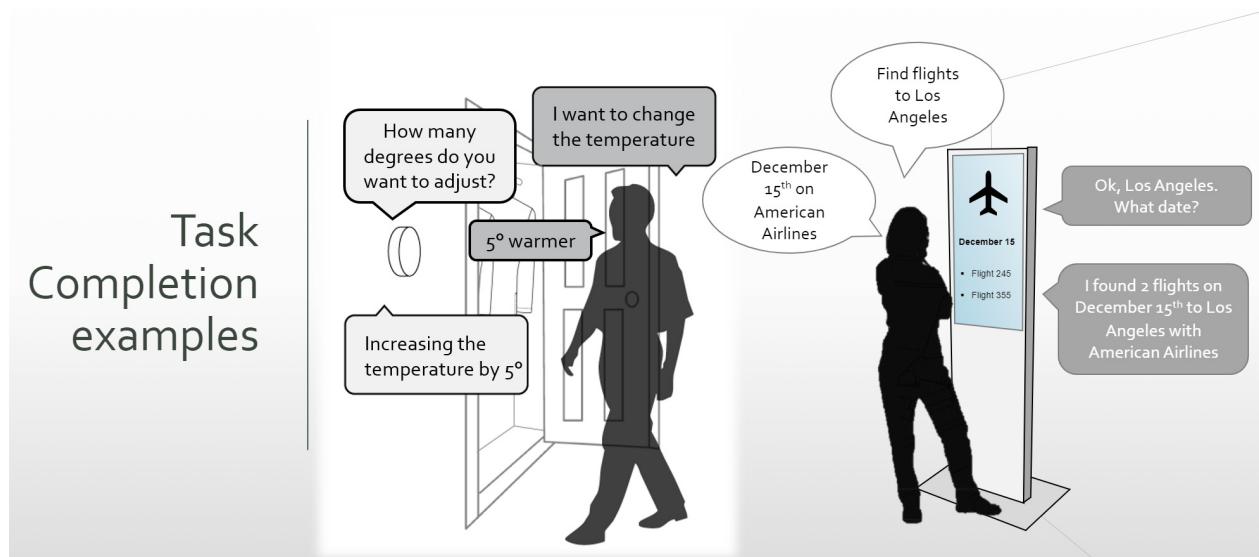
12/10/2019 • 2 minutes to read • [Edit Online](#)

Voice applications such as [Voice assistants](#) listen to users and take an action in response, often speaking back. They use [speech-to-text](#) to transcribe the user's speech, then take action on the natural language understanding of the text. This action frequently includes spoken output from the assistant generated with [text-to-speech](#). Devices connect to assistants with the Speech SDK's `DialogServiceConnector` object.

**Custom Commands (Preview)** is a streamlined solution for creating voice applications. It provides a unified authoring experience, an automatic hosting model, and relatively lower complexity versus other options like [Direct Line Speech](#). This simplification, however, comes with a reduction in flexibility. So, Custom Commands (Preview) is best suited for task completion or command-and-control scenarios. It's particularly well-matched for Internet of Things (IoT) and headless devices.

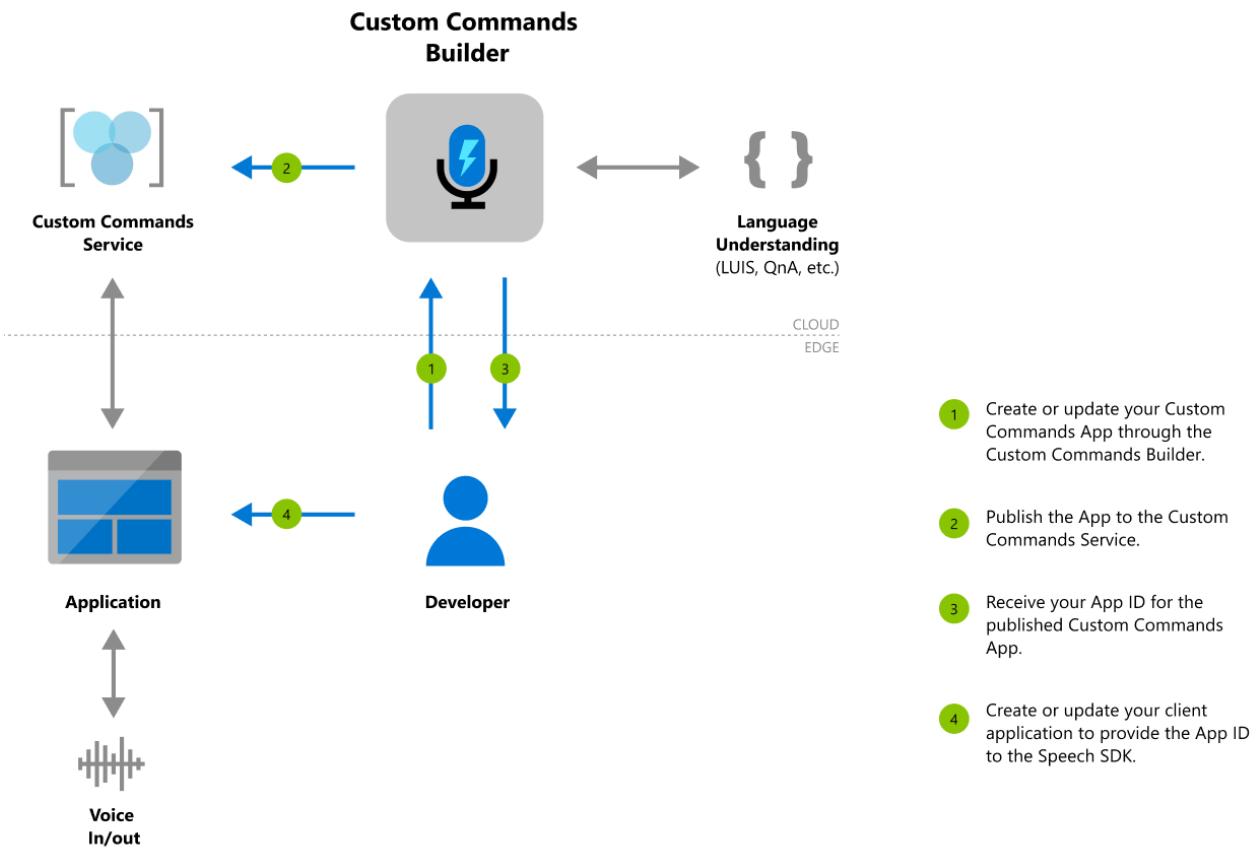
For complex conversational interaction and integration with other solutions like the [Virtual Assistant Solution](#) and [Enterprise Template](#) you're encouraged to use Direct Line Speech.

Good candidates for Custom Commands (Preview) have a fixed vocabulary with well-defined sets of variables. For example, home automation tasks, like controlling a thermostat, are ideal.



## Getting started with Custom Commands (Preview)

The first step for using Custom Commands (Preview) to make a voice application is to [get a speech subscription key](#) and access the Custom Commands (Preview) Builder on the [Speech Studio](#). From there, you can author a new Custom Commands (Preview) Application and publish it, after which an on-device application can communicate with it using the Speech SDK.



We offer quickstarts designed to have you running code in less than 10 minutes.

- [Create a Custom Commands \(Preview\) application](#)
- [Create a Custom Commands \(Preview\) application with parameters](#)
- [Connect to a Custom Commands \(Preview\) application with the Speech SDK, C#](#)

Once you are done with the quickstarts, explore our how-tos.

- [Add validations to Custom Command parameters](#)
- [Fulfill Commands on the client with the Speech SDK](#)
- [Add a confirmation to a Custom Command](#)
- [Add a one-step correction to a Custom Command](#)

## Next steps

- [Get a Speech service subscription key for free](#)
- [Go to the Speech Studio to try out Custom Commands](#)
- [Get the Speech SDK](#)

# About Direct Line Speech

12/4/2019 • 2 minutes to read • [Edit Online](#)

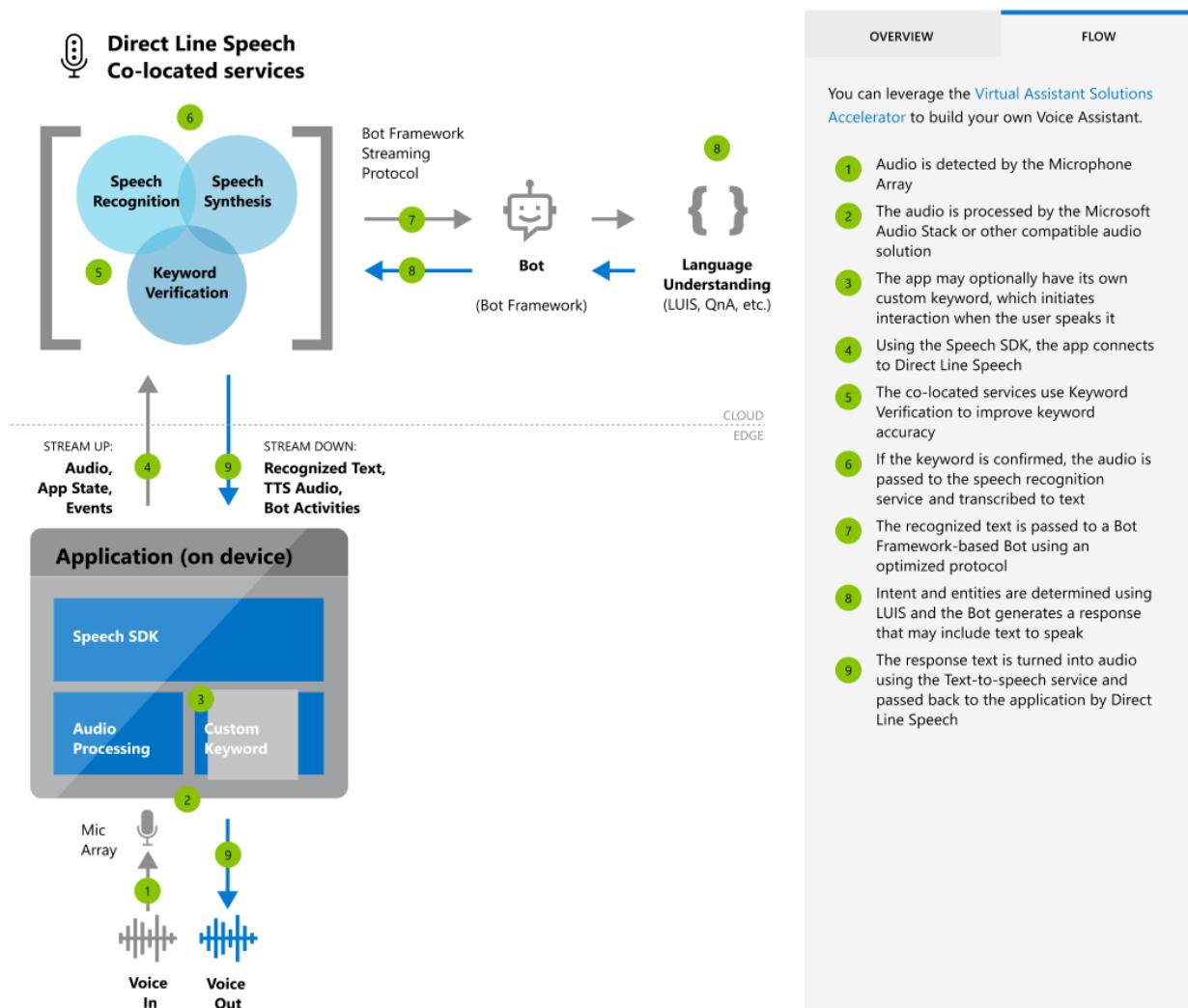
Voice assistants listen to users and take an action in response, often speaking back. They use [speech-to-text](#) to transcribe the user's speech, then take action on the natural language understanding of the text. This action frequently includes spoken output from the assistant generated with [text-to-speech](#). Devices connect to assistants with the Speech SDK's `DialogServiceConnector` object.

**Direct Line Speech** is a robust, end-to-end solution for creating a flexible, extensible voice assistant, powered by the Bot Framework and its Direct Line Speech channel, that is optimized for voice-in, voice-out interaction with bots.

Direct Line Speech offers the highest levels of customization and sophistication for voice assistants. It is well-suited to conversational scenarios that're open-ended, natural, or hybrids of these with task completion or command-and-control use. This high degree of flexibility comes with a greater complexity, and scenarios that are scoped to well-defined tasks using natural language input may want to consider [Custom Commands \(Preview\)](#) for a streamlined solution experience.

## Getting started with Direct Line Speech

The first steps for creating a voice assistant using Direct Line Speech are to [get a speech subscription key](#), create a new bot associated with that subscription, and connect the bot to the Direct Line Speech Channel.



For a complete, step-by-step guide on creating a simple voice assistant using Direct Line Speech, please see [the tutorial for speech-enabling your bot with the Speech SDK and the Direct Line Speech channel](#).

We also offer Quickstarts designed to have you running code in less than 10 minutes. This table includes a list of voice assistant quickstarts organized by language.

QUICKSTART	PLATFORM	API REFERENCE
C#, UWP	Windows	<a href="#">Browse</a>
Java	Windows, macOS, Linux	<a href="#">Browse</a>
Java	Android	<a href="#">Browse</a>

## Sample code

Sample code for creating a voice assistant is available on GitHub. These samples cover the client application for connecting to your assistant in several popular programming languages.

- [Voice assistant samples \(SDK\)](#)
- [Tutorial: Voice enable your assistant with the Speech SDK, C#](#)

## Customization

Voice assistants built using Speech service can use the full range of customization options available for [speech-to-text](#), [text-to-speech](#), and [custom keyword selection](#).

### NOTE

Customization options vary by language/locale (see [Supported languages](#)).

Direct Line Speech and its associated functionality for voice assistants are an ideal supplement to the [Virtual Assistant Solution and Enterprise Template](#). Though Direct Line Speech can work with any compatible bot, these resources provide a reusable baseline for high-quality conversational experiences as well as common supporting skills and models for getting started quickly.

## Reference docs

- [Speech SDK](#)
- [Azure Bot Service](#)

## Next steps

- [Get a Speech service subscription key for free](#)
- [Get the Speech SDK](#)
- [Create and deploy a basic bot](#)
- [Get the Virtual Assistant Solution and Enterprise Template](#)

# Quickstart: Create a voice assistant with the Speech SDK, UWP

12/4/2019 • 10 minutes to read • [Edit Online](#)

Quickstarts are also available for [speech recognition](#), [speech synthesis](#), and [speech translation](#).

In this article, you'll develop a C# Universal Windows Platform (UWP) application using the [Speech SDK](#). The program will connect to a previously authored and configured bot to enable a voice assistant experience from the client application. The application is built with the [Speech SDK NuGet Package](#) and Microsoft Visual Studio 2019 (any edition).

## NOTE

The Universal Windows Platform lets you develop apps that run on any device that supports Windows 10, including PCs, Xbox, Surface Hub, and other devices.

## Prerequisites

This quickstart requires:

- [Visual Studio 2019](#).
- An Azure subscription key for the Speech service. [Get one for free](#) or create it on the [Azure portal](#).
- A previously created bot configured with the [Direct Line Speech channel](#).

## NOTE

Please refer to [the list of supported regions for voice assistants](#) and ensure your resources are deployed in one of those regions.

## Optional: Get started fast

This quickstart will describe, step by step, how to make a client application to connect to your speech-enabled bot. If you prefer to dive right in, the complete, ready-to-compile source code used in this quickstart is available in the [Speech SDK Samples](#) under the `quickstart` folder.

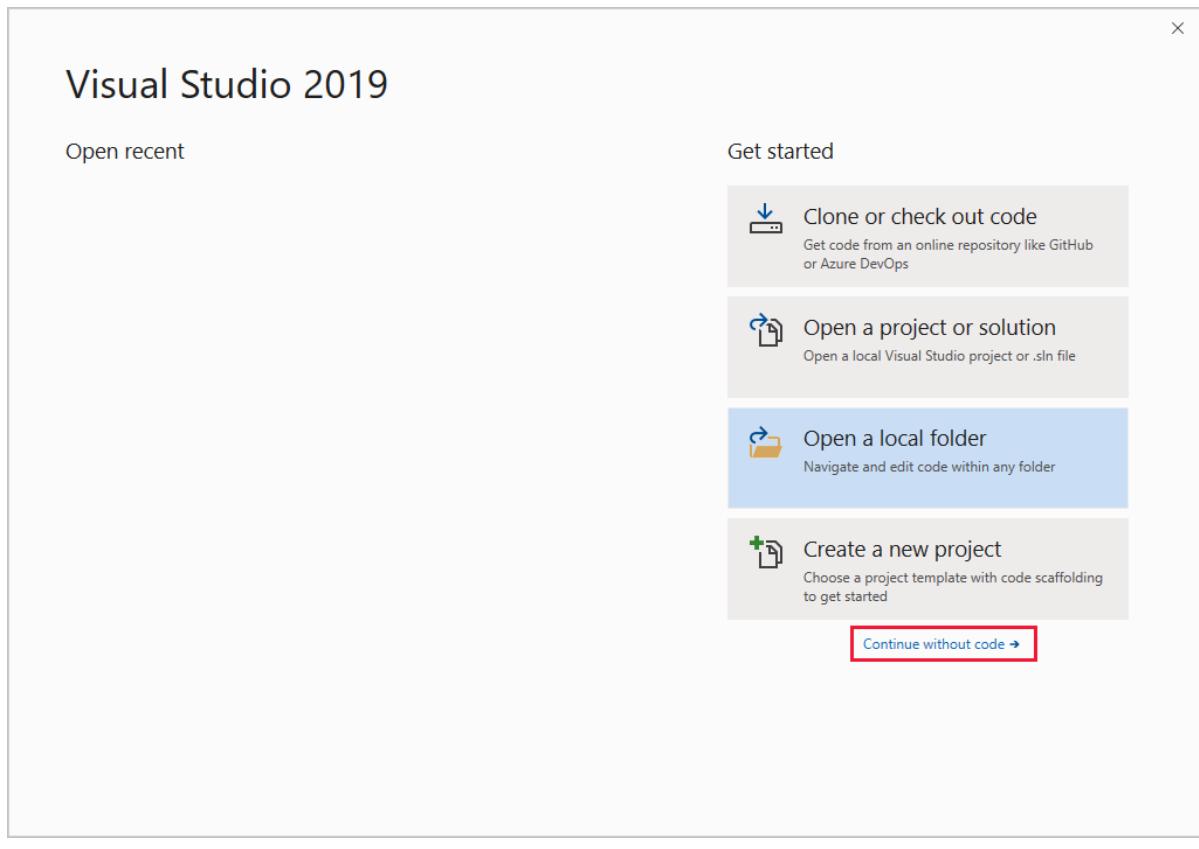
## Create a Visual Studio project

To create a Visual Studio project for Universal Windows Platform (UWP) development, you need to set up Visual Studio development options, create the project, select the target architecture, set up audio capture, and install the Speech SDK.

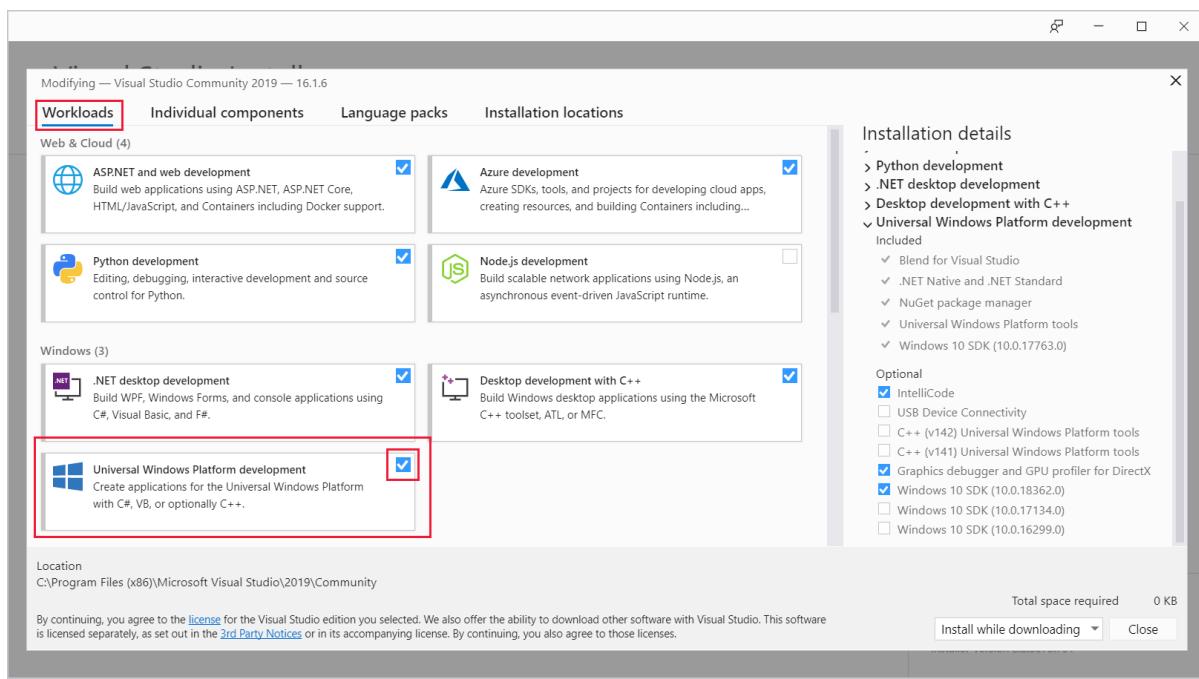
### Set up Visual Studio development options

To start, make sure you're set up correctly in Visual Studio for UWP development:

1. Open Visual Studio 2019 to display the **Start** window.



2. Select **Continue without code** to go to the Visual Studio IDE.
3. From the Visual Studio menu bar, select **Tools > Get Tools and Features** to open Visual Studio Installer and view the **Modifying** dialog box.

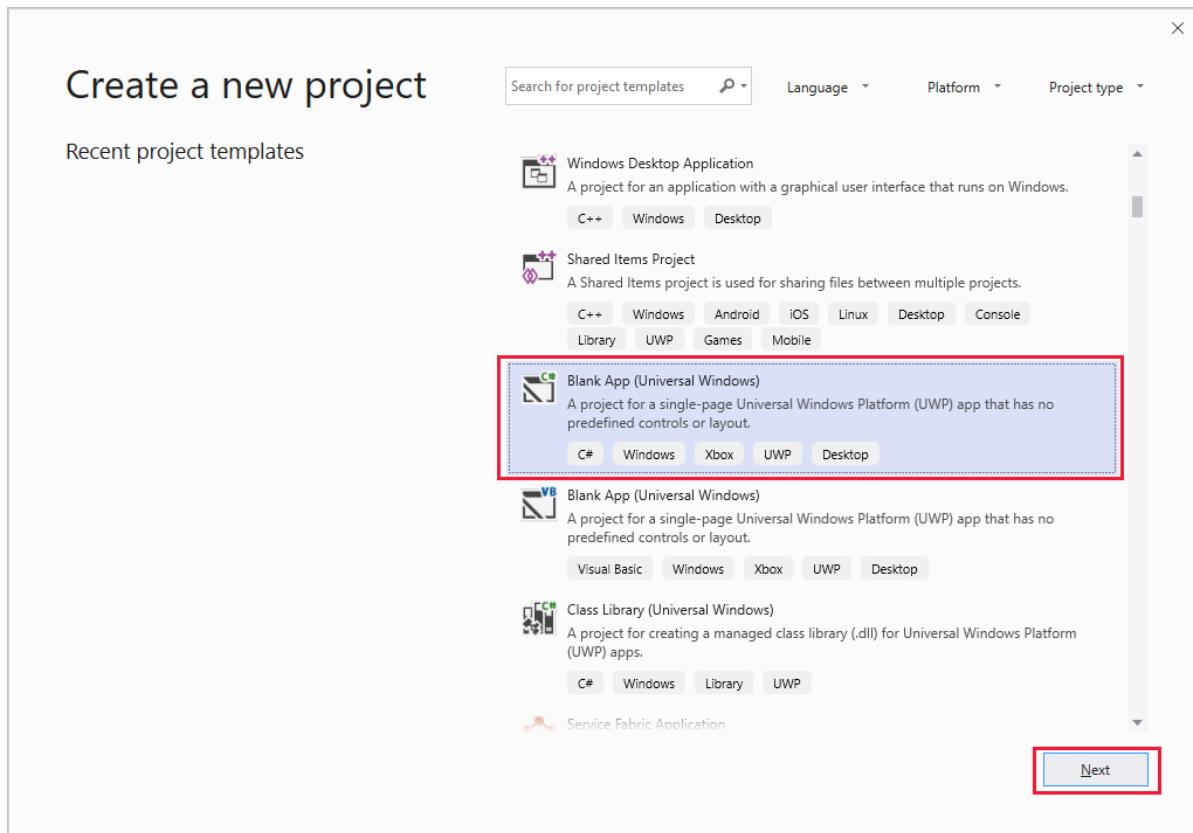


4. In the **Workloads** tab, under **Windows**, find the **Universal Windows Platform development** workload. If the check box next to that workload is already selected, close the **Modifying** dialog box, and go to step 6.
5. Select the **Universal Windows Platform development** check box, select **Modify**, and then in the **Before we get started** dialog box, select **Continue** to install the UWP development workload. Installation of the new feature may take a while.
6. Close Visual Studio Installer.

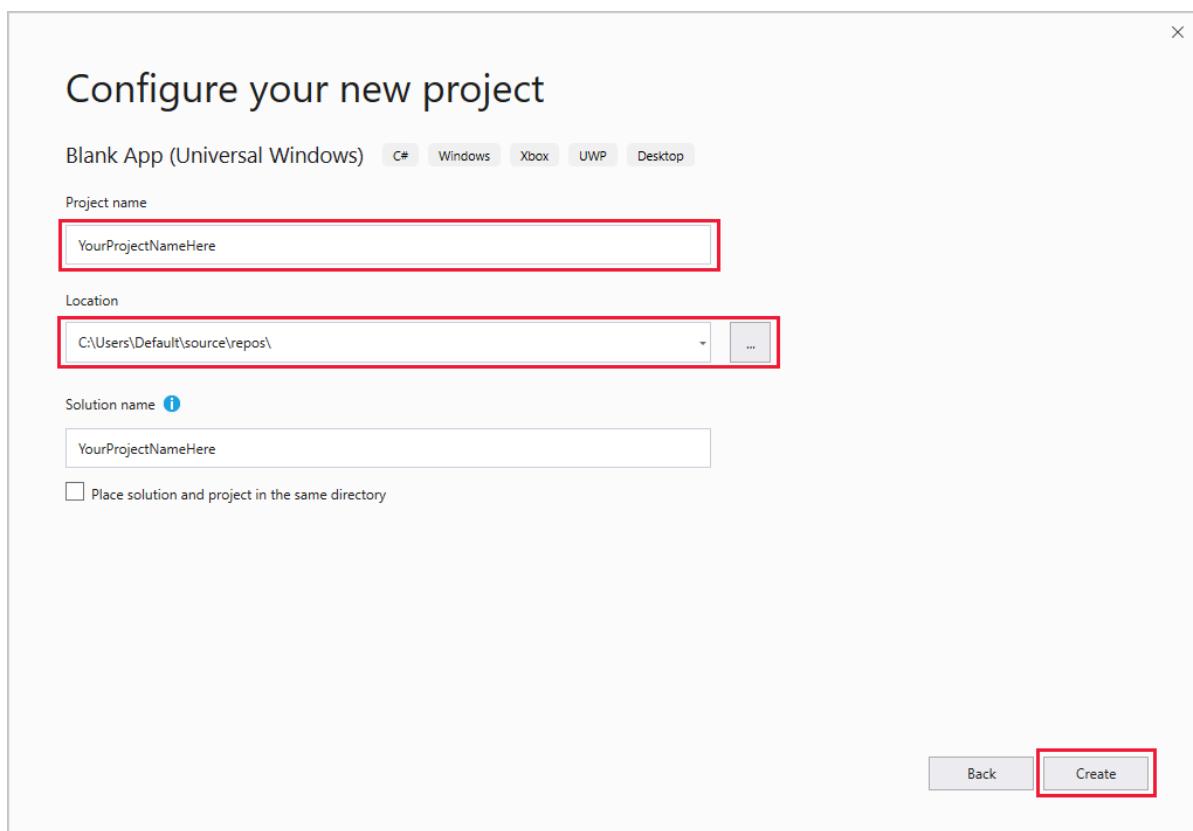
**Create the project and select the target architecture**

Next, create your project:

1. In the Visual Studio menu bar, choose **File > New > Project** to display the **Create a new project** window.

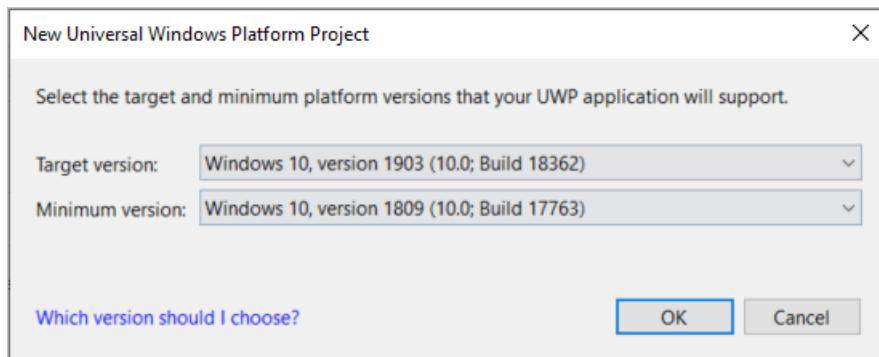


2. Find and select **Blank App (Universal Windows)**. Make sure that you select the C# version of this project type (as opposed to Visual Basic).
3. Select **Next** to display the **Configure your new project** screen.

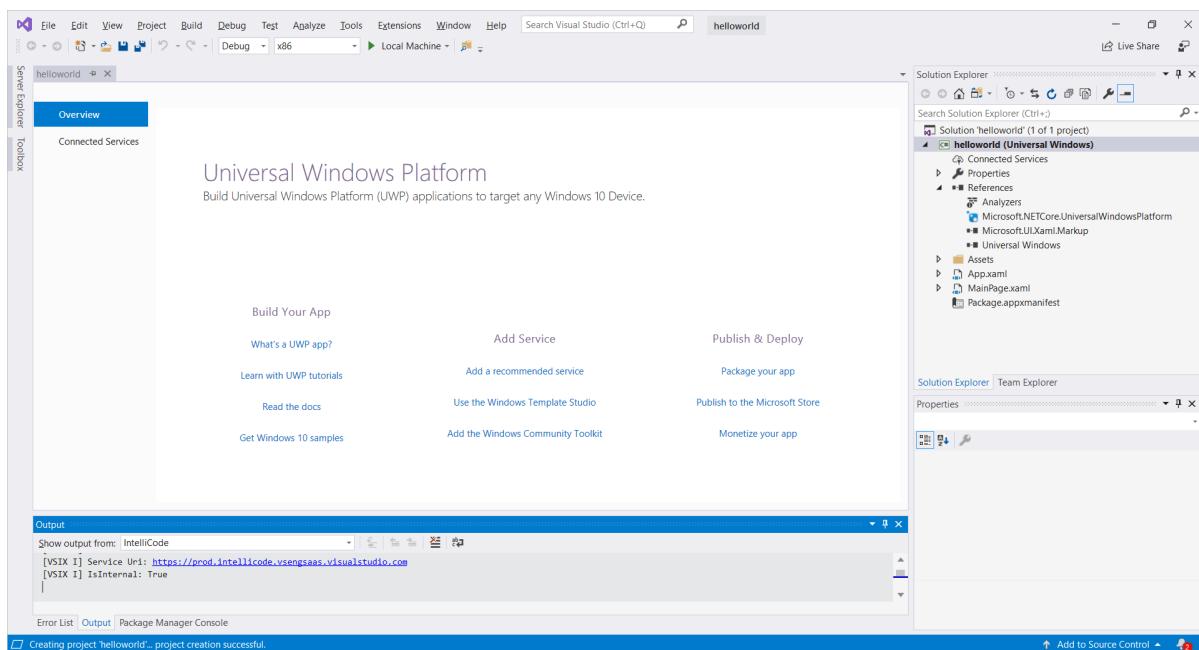


4. In **Project name**, enter `helloworld`.

- In **Location**, navigate to and select or create the folder to save your project in.
- Select **Create** to go to the **New Universal Windows Platform Project** window.



- In **Minimum version** (the second drop-down box), choose **Windows 10 Fall Creators Update (10.0; Build 16299)**, which is the minimum requirement for the Speech SDK.
- In **Target version** (the first drop-down box), choose a value identical to or later than the value in **Minimum version**.
- Select **OK**. You're returned to the Visual Studio IDE, with the new project created and visible in the **Solution Explorer** pane.



Now select your target platform architecture. In the Visual Studio toolbar, find the **Solution Platforms** drop-down box. (If you don't see it, choose **View > Toolbars > Standard** to display the toolbar containing **Solution Platforms**.) If you're running 64-bit Windows, choose **x64** in the drop-down box. 64-bit Windows can also run 32-bit applications, so you can choose **x86** if you prefer.

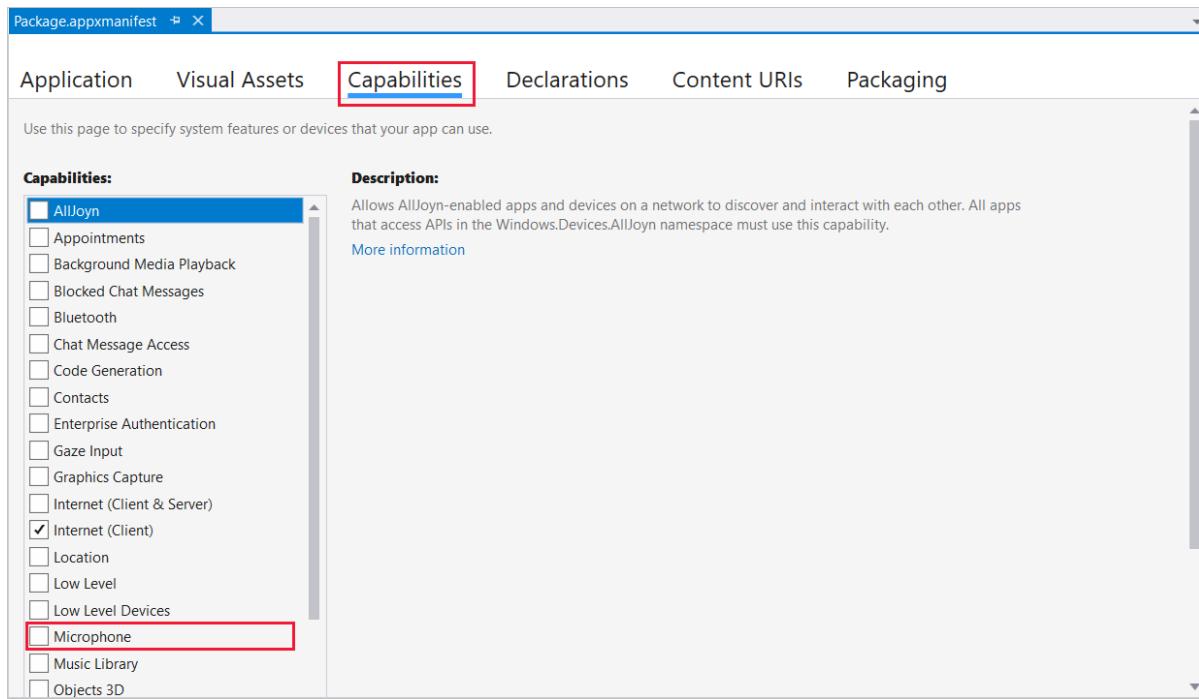
#### NOTE

The Speech SDK only supports Intel-compatible processors. ARM processors are currently not supported.

### Set up audio capture

Then allow the project to capture audio input:

- In **Solution Explorer**, double-click **Package.appxmanifest** to open the package application manifest.
- Select the **Capabilities** tab.

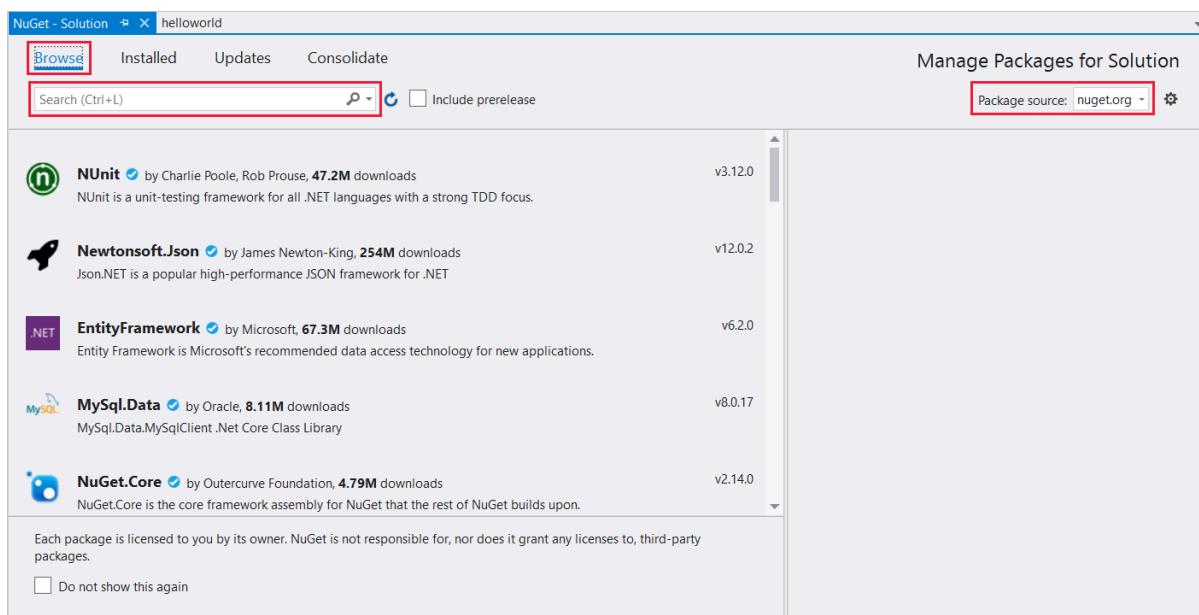


3. Select the box for the **Microphone** capability.
4. From the menu bar, choose **File > Save Package.appxmanifest** to save your changes.

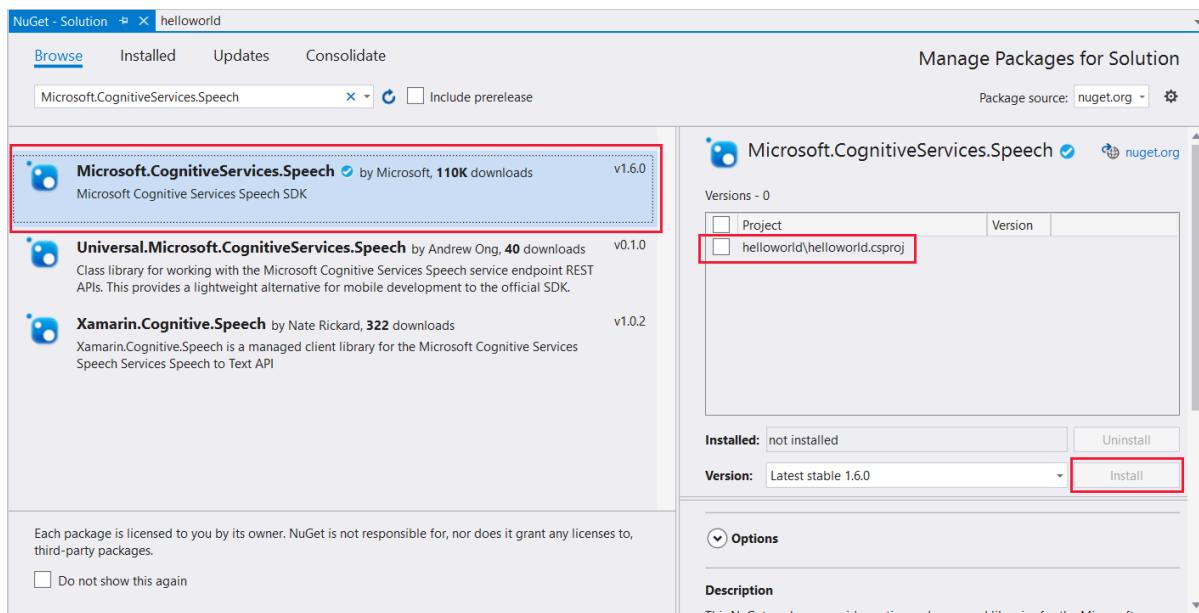
## Install the Speech SDK

Finally, install the [Speech SDK NuGet package](#), and reference the Speech SDK in your project:

1. In **Solution Explorer**, right-click your solution, and choose **Manage NuGet Packages for Solution** to go to the **NuGet - Solution** window.
2. Select **Browse**.



3. In **Package source**, choose **nuget.org**.
4. In the **Search** box, enter **Microsoft.CognitiveServices.Speech**, and then choose that package after it appears in the search results.



5. In the package status pane next to the search results, select your **helloworld** project.
6. Select **Install**.
7. In the **Preview Changes** dialog box, select **OK**.
8. In the **License Acceptance** dialog box, view the license, and then select **I Accept**. The package installation begins, and when installation is complete, the **Output** pane displays a message similar to the following text:  
`Successfully installed 'Microsoft.CognitiveServices.Speech 1.7.0' to helloworld .`

## Add sample code

Now add the XAML code that defines the user interface of the application, and add the C# code-behind implementation.

### XAML code

First, you'll create the application's user interface by adding the XAML code:

1. In **Solution Explorer**, open `MainPage.xaml`.
2. In the designer's XAML view, replace the entire contents with the following code snippet:

```

<Page
 x:Class="helloworld.MainPage"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:local="using:helloworld"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 mc:Ignorable="d"
 Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

 <Grid>
 <StackPanel Orientation="Vertical" HorizontalAlignment="Center"
 Margin="20,50,0,0" VerticalAlignment="Center" Width="800">
 <Button x:Name="EnableMicrophoneButton" Content="Enable Microphone"
 Margin="0,0,10,0" Click="EnableMicrophone_ButtonClicked"
 Height="35"/>
 <Button x:Name="ListenButton" Content="Talk to your bot"
 Margin="0,10,10,0" Click="ListenButton_ButtonClicked"
 Height="35"/>
 <StackPanel x:Name="StatusPanel" Orientation="Vertical"
 RelativePanel.AlignBottomWithPanel="True"
 RelativePanel.AlignRightWithPanel="True"
 RelativePanel.AlignLeftWithPanel="True">
 <TextBlock x:Name="StatusLabel" Margin="0,10,10,0"
 TextWrapping="Wrap" Text="Status:" FontSize="20"/>
 <Border x:Name="StatusBorder" Margin="0,0,0,0">
 <ScrollViewer VerticalScrollMode="Auto"
 VerticalScrollBarVisibility="Auto" MaxHeight="200">
 <!-- Use LiveSetting to enable screen readers to announce
 the status update. -->
 <TextBlock
 x:Name="StatusBlock" FontWeight="Bold"
 AutomationProperties.LiveSetting="Assertive"
 MaxWidth="{Binding ElementName=Splitter, Path=ActualWidth}"
 Margin="10,10,10,20" TextWrapping="Wrap" />
 </ScrollViewer>
 </Border>
 </StackPanel>
 </StackPanel>
 <MediaElement x:Name="mediaElement"/>
 </Grid>
</Page>

```

The Design view is updated to show the application's user interface.

## C# code-behind source

Then you add the code-behind source so that the application works as expected. The code-behind source includes:

- `using` statements for the `Speech` and `Speech.Dialog` namespaces
- A simple implementation to ensure microphone access, wired to a button handler
- Basic UI helpers to present messages and errors in the application
- A landing point for the initialization code path that will be populated later
- A helper to play back text-to-speech (without streaming support)
- An empty button handler to start listening that will be populated later

To add the code-behind source, follow these steps:

1. In **Solution Explorer**, open the code-behind source file `MainPage.xaml.cs`. (It's grouped under `MainPage.xaml`.)
2. Replace the file's contents with the following code snippet:

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Dialog;
using System;
using System.Diagnostics;
using System.IO;
using System.Text;
using Windows.Foundation;
using Windows.Storage.Streams;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;

namespace helloworld
{
 public sealed partial class MainPage : Page
 {
 private DialogServiceConnector connector;

 private enum NotifyType
 {
 StatusMessage,
 ErrorMessage
 };

 public MainPage()
 {
 this.InitializeComponent();
 }

 private async void EnableMicrophone_ButtonClicked(
 object sender, RoutedEventArgs e)
 {
 bool isMicAvailable = true;
 try
 {
 var mediaCapture = new Windows.Media.Capture.MediaCapture();
 var settings =
 new Windows.Media.Capture.MediaCaptureInitializationSettings();
 settings.StreamingCaptureMode =
 Windows.Media.Capture.StreamingCaptureMode.Audio;
 await mediaCapture.InitializeAsync(settings);
 }
 catch (Exception)
 {
 isMicAvailable = false;
 }
 if (!isMicAvailable)
 {
 await Windows.System.Launcher.LaunchUriAsync(
 new Uri("ms-settings:privacy-microphone"));
 }
 else
 {
 NotifyUser("Microphone was enabled", NotifyType.StatusMessage);
 }
 }

 private void NotifyUser(
 string strMessage, NotifyType type = NotifyType.StatusMessage)
 {
 // If called from the UI thread, then update immediately.
 // Otherwise, schedule a task on the UI thread to perform the update.
 if (Dispatcher.HasThreadAccess)
 {
 UpdateStatus(strMessage, type);
 }
 else
 {

```

```

 var task = Dispatcher.RunAsync(
 Windows.UI.Core.CoreDispatcherPriority.Normal,
 () => UpdateStatus(strMessage, type));
 }

 private void UpdateStatus(string strMessage, NotifyType type)
 {
 switch (type)
 {
 case NotifyType.StatusMessage:
 StatusBorder.Background = new SolidColorBrush(
 Windows.UI.Colors.Green);
 break;
 case NotifyType.ErrorMessage:
 StatusBorder.Background = new SolidColorBrush(
 Windows.UI.Colors.Red);
 break;
 }
 StatusBlock.Text += string.IsNullOrEmpty(StatusBlock.Text)
 ? strMessage : "\n" + strMessage;

 if (!string.IsNullOrEmpty(StatusBlock.Text))
 {
 StatusBorder.Visibility = Visibility.Visible;
 StatusPanel.Visibility = Visibility.Visible;
 }
 else
 {
 StatusBorder.Visibility = Visibility.Collapsed;
 StatusPanel.Visibility = Visibility.Collapsed;
 }
 // Raise an event if necessary to enable a screen reader
 // to announce the status update.
 var peer =
Windows.UI.Xaml.Automation.Peers.FrameworkElementAutomationPeer.FromElement(StatusBlock);
 if (peer != null)
 {
 peer.RaiseAutomationEvent(
 Windows.UI.Xaml.Automation.Peers.AutomationEvents.LiveRegionChanged);
 }
 }

 // Waits for and accumulates all audio associated with a given
 // PullAudioOutputStream and then plays it to the MediaElement. Long spoken
 // audio will create extra latency and a streaming playback solution
 // (that plays audio while it continues to be received) should be used --
 // see the samples for examples of this.
 private void SynchronouslyPlayActivityAudio(
 PullAudioOutputStream activityAudio)
 {
 var playbackStreamWithHeader = new MemoryStream();
 playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("RIFF"), 0, 4); // ChunkID
 playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); // ChunkSize:
max
 playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("WAVE"), 0, 4); // Format
 playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("fmt "), 0, 4); // Subchunk1ID
 playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 4); // Subchunk1Size: PCM
 playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // AudioFormat: PCM
 playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // NumChannels: mono
 playbackStreamWithHeader.Write(BitConverter.GetBytes(16000), 0, 4); // SampleRate: 16kHz
 playbackStreamWithHeader.Write(BitConverter.GetBytes(32000), 0, 4); // ByteRate
 playbackStreamWithHeader.Write(BitConverter.GetBytes(2), 0, 2); // BlockAlign
 playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 2); // BitsPerSample: 16-bit
 playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("data"), 0, 4); // Subchunk2ID
 playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); //
Subchunk2Size
 byte[] nullBuffer = new byte[2056];

```

```

 uint lastRead = 0;
 do
 {
 lastRead = activityAudio.Read(pullBuffer);
 playbackStreamWithHeader.Write(pullBuffer, 0, (int)lastRead);
 }
 while (lastRead == pullBuffer.Length);

 var task = Dispatcher.RunAsync(
 Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
 {
 mediaElement.SetSource(
 playbackStreamWithHeader.AsRandomAccessStream(), "audio/wav");
 mediaElement.Play();
 });
 }

 private void InitializeDialogServiceConnector()
 {
 // New code will go here
 }

 private async void ListenButton_ButtonClicked(
 object sender, RoutedEventArgs e)
 {
 // New code will go here
 }
}
}

```

- Add the following code snippet to the method body of `InitializeDialogServiceConnector`. This code creates the `DialogServiceConnector` with your subscription information.

```

// Create a BotFrameworkConfig by providing a Speech service subscription key
// the RecoLanguage property is optional (default en-US)
const string speechSubscriptionKey = "YourSpeechSubscriptionKey"; // Your subscription key
const string region = "YourServiceRegion"; // Your subscription service region.

var botConfig = BotFrameworkConfig.FromSubscription(speechSubscriptionKey, region);
botConfig SetProperty(PropertyId.SpeechServiceConnection_RecoLanguage, "en-US");
connector = new DialogServiceConnector(botConfig);

```

#### NOTE

Please refer to the [list of supported regions for voice assistants](#) and ensure your resources are deployed in one of those regions.

#### NOTE

For information on configuring your bot and retrieving a channel secret, see the Bot Framework documentation for the [Direct Line Speech channel](#).

- Replace the strings `YourChannelSecret`, `YourSpeechSubscriptionKey`, and `YourServiceRegion` with your own values for your bot, speech subscription, and `region`.
- Append the following code snippet to the end of the method body of `InitializeDialogServiceConnector`. This code sets up handlers for events relied on by `DialogServiceConnector` to communicate its bot activities, speech recognition results, and other information.

```

// ActivityReceived is the main way your bot will communicate with the client
// and uses bot framework activities
connector.ActivityReceived += (sender, activityReceivedEventArgs) =>
{
 NotifyUser(
 $"Activity received, hasAudio={activityReceivedEventArgs.HasAudio} activity={activityReceivedEventArgs.Activity}");

 if (activityReceivedEventArgs.HasAudio)
 {
 SynchronouslyPlayActivityAudio(activityReceivedEventArgs.Audio);
 }
};

// Canceled will be signaled when a turn is aborted or experiences an error condition
connector.Canceled += (sender, canceledEventArgs) =>
{
 NotifyUser($"Canceled, reason={canceledEventArgs.Reason}");
 if (canceledEventArgs.Reason == CancellationReason.Error)
 {
 NotifyUser(
 $"Error: code={canceledEventArgs.ErrorCode}, details={canceledEventArgs.ErrorDetails}");
 }
};

// Recognizing (not 'Recognized') will provide the intermediate recognized text
// while an audio stream is being processed
connector.Recognizing += (sender, recognitionEventArgs) =>
{
 NotifyUser($"Recognizing! in-progress text={recognitionEventArgs.Result.Text}");
};

// Recognized (not 'Recognizing') will provide the final recognized text
// once audio capture is completed
connector.Recognized += (sender, recognitionEventArgs) =>
{
 NotifyUser($"Final speech-to-text result: '{recognitionEventArgs.Result.Text}'");
};

// SessionStarted will notify when audio begins flowing to the service for a turn
connector.SessionStarted += (sender, sessionEventArgs) =>
{
 NotifyUser($"Now Listening! Session started, id={sessionEventArgs.SessionId}");
};

// SessionStopped will notify when a turn is complete and
// it's safe to begin listening again
connector.SessionStopped += (sender, sessionEventArgs) =>
{
 NotifyUser($"Listening complete. Session ended, id={sessionEventArgs.SessionId}");
};

```

6. Add the following code snippet to the body of the `ListenButton_ButtonClicked` method in the `MainPage` class. This code sets up `DialogServiceConnector` to listen, since you already established the configuration and registered the event handlers.

```
if (connector == null)
{
 InitializeDialogServiceConnector();
 // Optional step to speed up first interaction: if not called,
 // connection happens automatically on first use
 var connectTask = connector.ConnectAsync();
}

try
{
 // Start sending audio to your speech-enabled bot
 var listenTask = connector.ListenOnceAsync();

 // You can also send activities to your bot as JSON strings --
 // Microsoft.Bot.Schema can simplify this
 string speakActivity =
 @"{""type"":""message"" , ""text"":""Greeting Message"" , ""speak"":""Hello there!""}";
 await connector.SendActivityAsync(speakActivity);

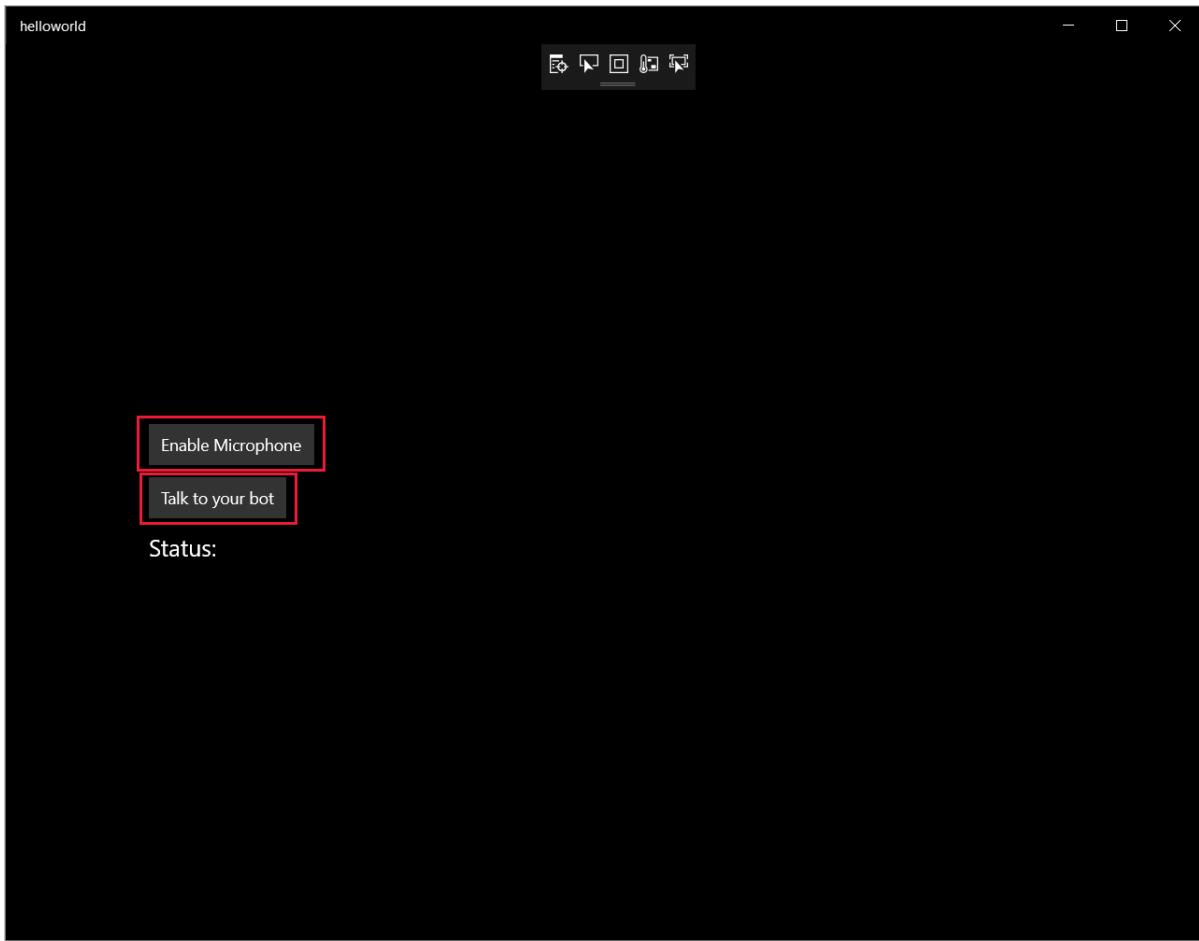
}
catch (Exception ex)
{
 NotifyUser($"Exception: {ex.ToString()}", NotifyType.ErrorMessage);
}
```

7. From the menu bar, choose **File** > **Save All** to save your changes.

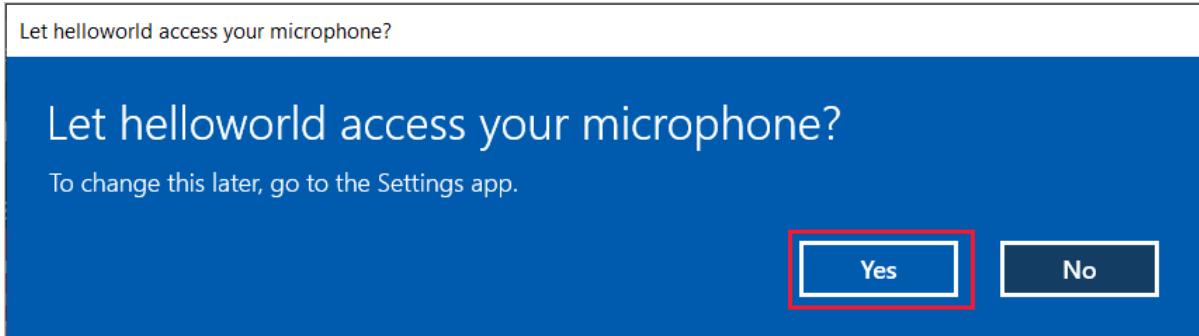
## Build and run the application

Now you are ready to build and test your application.

1. From the menu bar, choose **Build** > **Build Solution** to build the application. The code should compile without errors now.
2. Choose **Debug** > **Start Debugging** (or press **F5**) to start the application. The **helloworld** window appears.



3. Select **Enable Microphone**, and when the access permission request pops up, select **Yes**.



4. Select **Talk to your bot**, and speak an English phrase or sentence into your device's microphone. Your speech is transmitted to the Direct Line Speech channel and transcribed to text, which appears in the window.

## Next steps

[Create and deploy a basic bot](#)

## See also

- [About voice assistants](#)
- [Get a Speech service subscription key for free](#)
- [Custom keywords](#)
- [Connect Direct Line Speech to your bot](#)
- [Explore C# samples on GitHub](#)

# Quickstart: Create a voice assistant with the Speech SDK, Java (Preview)

12/4/2019 • 11 minutes to read • [Edit Online](#)

Quickstarts are also available for [speech-to-text](#), [text-to-speech](#), and [speech translation](#).

In this article, you create a Java console application by using the [Azure Cognitive Services Speech SDK](#). The application connects to a previously authored bot configured to use the Direct Line Speech channel, sends a voice request, and returns a voice response activity (if configured). The application is built with the Speech SDK Maven package and the Eclipse Java IDE on Windows, Ubuntu Linux, or on macOS. It runs on a 64-bit Java 8 runtime environment (JRE).

## Prerequisites

This quickstart requires:

- Operating system: Windows (64-bit), Ubuntu Linux 16.04/18.04 (64-bit), or macOS 10.13 or later.
- [Eclipse Java IDE](#).
- [Java 8 or JDK 8](#).
- An Azure subscription key for the Speech service. [Get one for free](#) or create it in the [Azure portal](#).
- A preconfigured bot created by using Bot Framework version 4.2 or above. The bot needs to subscribe to the new Direct Line Speech channel to receive voice inputs.

### NOTE

Please refer to the [list of supported regions for voice assistants](#) and ensure your resources are deployed in one of those regions.

If you're running Ubuntu 16.04/18.04, make sure these dependencies are installed before you start Eclipse:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libasound2 wget
```

If you're running Windows (64-bit), make sure you installed the Microsoft Visual C++ Redistributable for your platform:

- [Download Microsoft Visual C++ Redistributable for Visual Studio 2017](#)

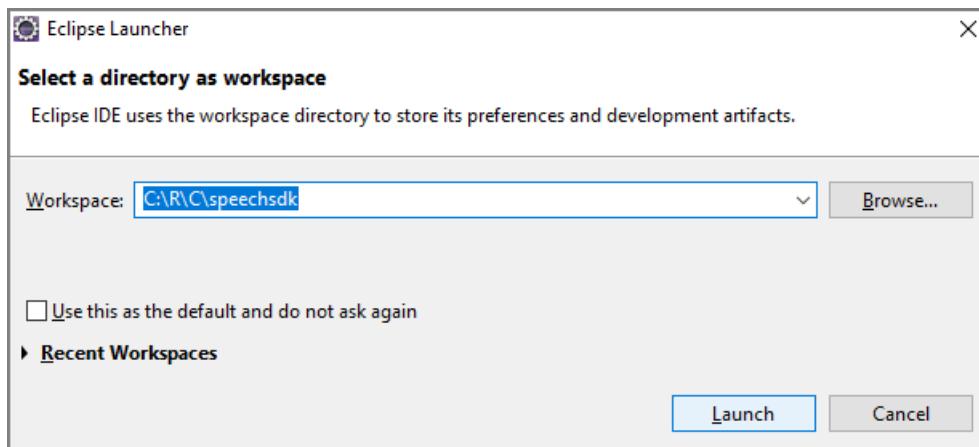
## Optional: Get started fast

This quickstart describes, step by step, how to make a simple client application to connect to your speech-enabled bot. If you want to dive right in, the complete, ready-to-compile source code used in this quickstart is available in the [Speech SDK samples](#) under the `quickstart` folder.

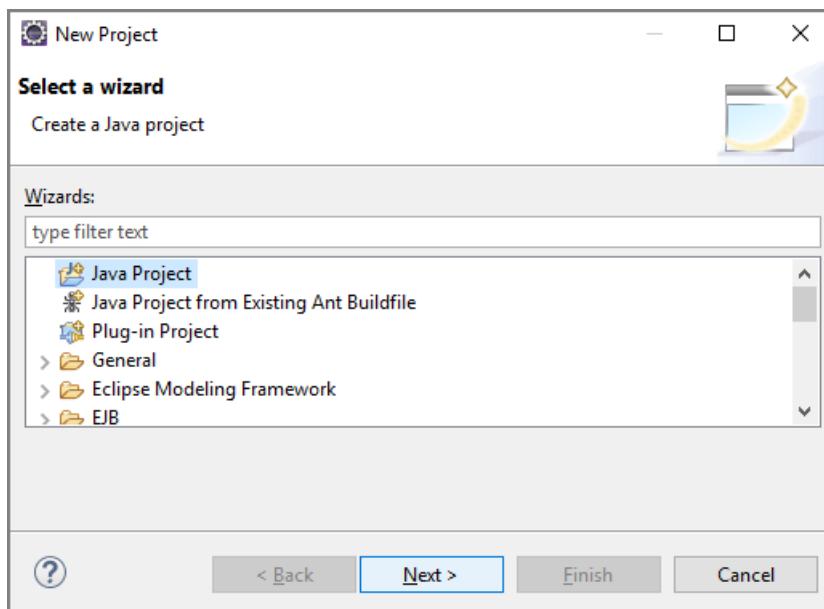
## Create and configure project

1. Start Eclipse.

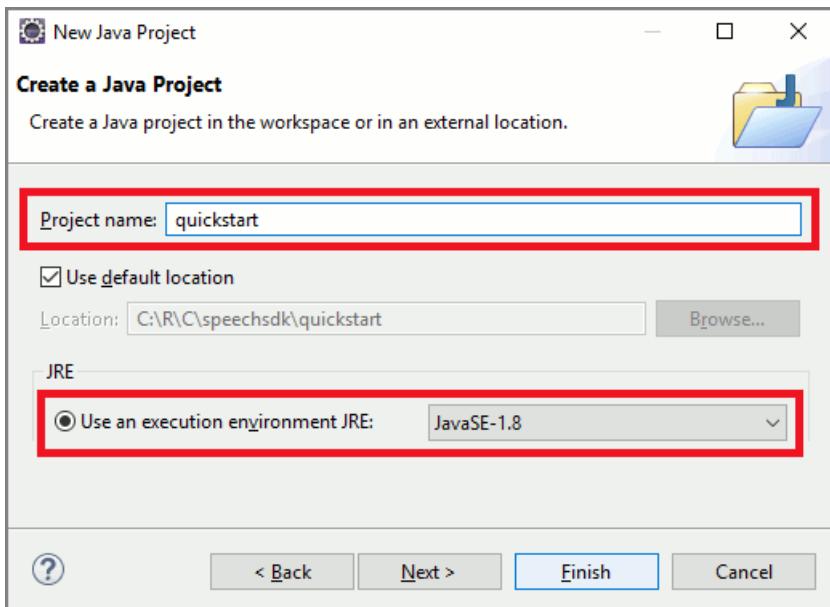
2. In the Eclipse Launcher, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.



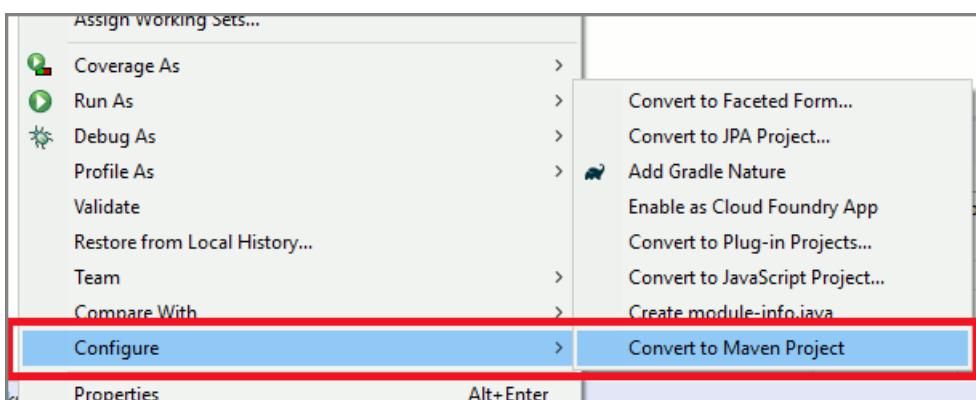
3. In a moment, the main window of the Eclipse IDE appears. Close the **Welcome** screen if one is present.
4. From the Eclipse menu bar, create a new project by choosing **File > New > Project**.
5. The **New Project** dialog box appears. Select **Java Project**, and select **Next**.



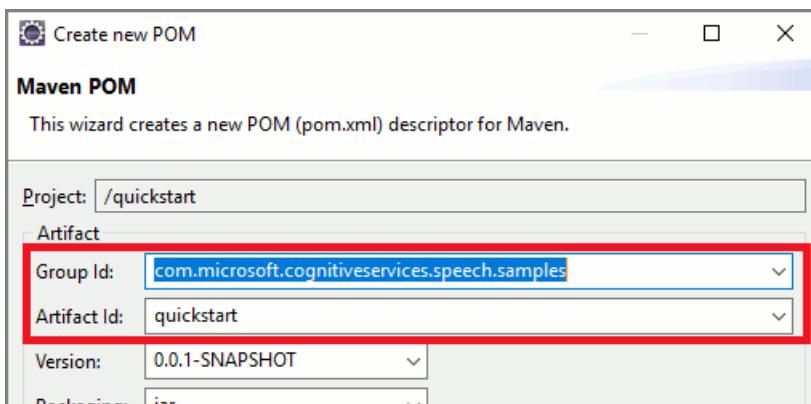
6. The **New Java Project** wizard starts. In the **Project name** field, enter **quickstart**, and choose **JavaSE-1.8** as the execution environment. Select **Finish**.



7. If the **Open Associated Perspective?** window appears, select **Open Perspective**.
8. In the **Package explorer**, right-click the **quickstart** project. Choose **Configure > Convert to Maven Project** from the context menu.



9. The **Create new POM** window appears. In the **Group Id** field, enter `com.microsoft.cognitiveservices.speech.samples`, and in the **Artifact Id** field, enter `quickstart`. Then select **Finish**.



10. Open the `pom.xml` file and edit it.

- At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK, as shown here:

```

<repositories>
 <repository>
 <id>maven-cognitiveservices-speech</id>
 <name>Microsoft Cognitive Services Speech Maven Repository</name>
 <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
 </repository>
</repositories>

```

- Also add a `dependencies` element, with the Speech SDK version 1.7.0 as a dependency:

```

<dependencies>
 <dependency>
 <groupId>com.microsoft.cognitiveservices.speech</groupId>
 <artifactId>client-sdk</artifactId>
 <version>1.7.0</version>
 </dependency>
</dependencies>

```

- Save the changes.

Additionally, to enable logging, update the `pom.xml` file to include the following dependency:

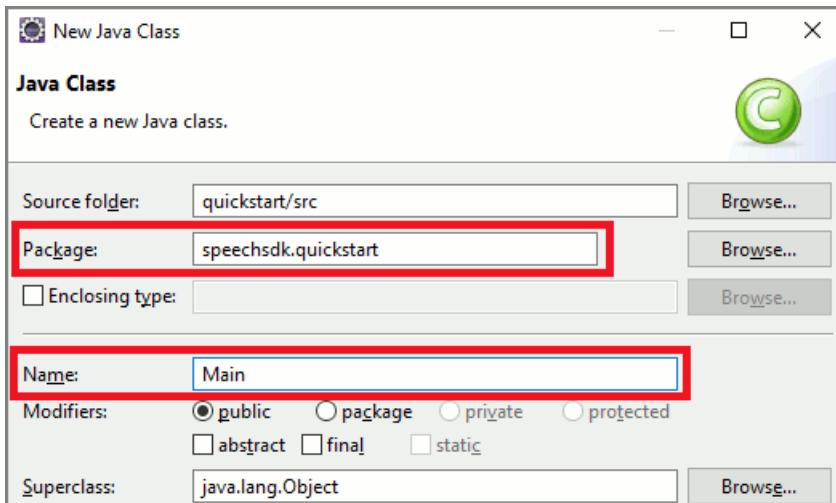
```

<dependency>
 <groupId>org.slf4j</groupId>
 <artifactId>slf4j-simple</artifactId>
 <version>1.7.5</version>
</dependency>

```

## Add sample code

- To add a new empty class to your Java project, select **File > New > Class**.
- In the **New Java Class** window, enter `speechsdk.quickstart` in the **Package** field and `Main` in the **Name** field.



- Open the newly created `Main` class, and replace the contents of the `Main.java` file with the following starting code:

```

package speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import com.microsoft.cognitiveservices.speech.audio.PullAudioOutputStream;
import com.microsoft.cognitiveservices.speech.dialog.DialogServiceConfig;
import com.microsoft.cognitiveservices.speech.dialog.DialogServiceConnector;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.SourceDataLine;
import java.io.InputStream;

public class Main {
 final Logger log = LoggerFactory.getLogger(Main.class);

 public static void main(String[] args) {
 // New code will go here
 }

 private void playAudioStream(PullAudioOutputStream audio) {
 ActivityAudioStream stream = new ActivityAudioStream(audio);
 final ActivityAudioStream.ActivityAudioFormat audioFormat = stream.getActivityAudioFormat();
 final AudioFormat format = new AudioFormat(
 AudioFormat.Encoding.PCM_SIGNED,
 audioFormat.getSamplesPerSecond(),
 audioFormat.getBitsPerSample(),
 audioFormat.getChannels(),
 audioFormat.getFrameSize(),
 audioFormat.getSamplesPerSecond(),
 false);
 try {
 int bufferSize = format.getFrameSize();
 final byte[] data = new byte[bufferSize];

 SourceDataLine.Info info = new DataLine.Info(SourceDataLine.class, format);
 SourceDataLine line = (SourceDataLine) AudioSystem.getLine(info);
 line.open(format);

 if (line != null) {
 line.start();
 int nBytesRead = 0;
 while (nBytesRead != -1) {
 nBytesRead = stream.read(data);
 if (nBytesRead != -1) {
 line.write(data, 0, nBytesRead);
 }
 }
 line.drain();
 line.stop();
 line.close();
 }
 stream.close();
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}

```

4. In the `main` method, you first configure your `DialogServiceConfig` and use it to create a `DialogServiceConnector` instance. This instance connects to the Direct Line Speech channel to interact with your bot. An `AudioConfig` instance is also used to specify the source for audio input. In this example, the

default microphone is used with `AudioConfig.fromDefaultMicrophoneInput()`.

- Replace the string `YourSubscriptionKey` with your subscription key, which you can get from [this website](#).
- Replace the string `YourServiceRegion` with the [region](#) associated with your subscription.

#### NOTE

Please refer to the [list of supported regions for voice assistants](#) and ensure your resources are deployed in one of those regions.

```
final String subscriptionKey = "YourSubscriptionKey"; // Your subscription key
final String region = "YourServiceRegion"; // Your speech subscription service region
final BotFrameworkConfig botConfig = BotFrameworkConfig.fromSubscription(subscriptionKey, region);

// Configure audio input from a microphone.
final AudioConfig audioConfig = AudioConfig.fromDefaultMicrophoneInput();

// Create a DialogServiceConnector instance.
final DialogServiceConnector connector = new DialogServiceConnector(botConfig, audioConfig);
```

5. The connector `DialogServiceConnector` relies on several events to communicate its bot activities, speech recognition results, and other information. Add these event listeners next.

```
// Recognizing will provide the intermediate recognized text while an audio stream is being processed.
connector.recognizing.addEventListener((o, speechRecognitionEventArgs) -> {
 log.info("Recognizing speech event text: {}", speechRecognitionEventArgs.getResult().getText());
});

// Recognized will provide the final recognized text once audio capture is completed.
connector.recognized.addEventListener((o, speechRecognitionEventArgs) -> {
 log.info("Recognized speech event reason text: {}", speechRecognitionEventArgs.getResult().getText());
});

// SessionStarted will notify when audio begins flowing to the service for a turn.
connector.sessionStarted.addEventListener((o, sessionEventArgs) -> {
 log.info("Session Started event id: {} ", sessionEventArgs.getSessionId());
});

// SessionStopped will notify when a turn is complete and it's safe to begin listening again.
connector.sessionStopped.addEventListener((o, sessionEventArgs) -> {
 log.info("Session stopped event id: {} ", sessionEventArgs.getSessionId());
});

// Canceled will be signaled when a turn is aborted or experiences an error condition.
connector.canceled.addEventListener((o, canceledEventArgs) -> {
 log.info("Canceled event details: {}", canceledEventArgs.getErrorDetails());
 connector.disconnectAsync();
});

// ActivityReceived is the main way your bot will communicate with the client and uses Bot Framework
// activities.
connector.activityReceived.addEventListener((o, activityEventArgs) -> {
 final String act = activityEventArgs.getActivity().serialize();
 log.info("Received activity {} audio", activityEventArgs.hasAudio() ? "with" : "without");
 if (activityEventArgs.hasAudio()) {
 playAudioStream(activityEventArgs.getAudio());
 }
});
```

6. Connect `DialogServiceConnector` to Direct Line Speech by invoking the `connectAsync()` method. To test your bot, you can invoke the `listenOnceAsync` method to send audio input from your microphone. Additionally, you can also use the `sendActivityAsync` method to send a custom activity as a serialized string. These custom activities can provide additional data that your bot uses in the conversation.

```
connector.connectAsync();
// Start listening.
System.out.println("Say something ...");
connector.listenOnceAsync();

// connector.sendActivityAsync(...)
```

7. Save changes to the `Main` file.
8. To support response playback, add an additional class that transforms the `PullAudioOutputStream` object returned from the `getAudio()` API to a Java `InputStream` for ease of handling. This `ActivityAudioStream` is a specialized class that handles audio response from the Direct Line Speech channel. It provides accessors to fetch audio format information that's required for handling playback. For that, select **File > New > Class**.
9. In the **New Java Class** window, enter `speechsdk.quickstart` in the **Package** field and `ActivityAudioStream` in the **Name** field.
10. Open the newly created `ActivityAudioStream` class, and replace it with the following code:

```
package com.speechsdk.quickstart;

import com.microsoft.cognitiveservices.speech.audio.PullAudioOutputStream;

import java.io.IOException;
import java.io.InputStream;

public final class ActivityAudioStream extends InputStream {

 /**
 * The number of samples played per second (16 kHz).
 */
 public static final long SAMPLE_RATE = 16000;
 /**
 * The number of bits in each sample of a sound that has this format (16 bits).
 */
 public static final int BITS_PER_SECOND = 16;
 /**
 * The number of audio channels in this format (1 for mono).
 */
 public static final int CHANNELS = 1;
 /**
 * The number of bytes in each frame of a sound that has this format (2).
 */
 public static final int FRAME_SIZE = 2;

 /**
 * Reads up to a specified maximum number of bytes of data from the audio
 * stream, putting them into the given byte array.
 *
 * @param b the buffer into which the data is read
 * @param off the offset, from the beginning of array <code>b</code>, at which
 * the data will be written
 * @param len the maximum number of bytes to read
 * @return the total number of bytes read into the buffer, or -1 if there
 * is no more data because the end of the stream has been reached
 */
 @Override
 public int read(byte[] b, int off, int len) {
 byte[] tempBuffer = new byte[len];
```

```

 int n = (int) this.pullStreamImpl.read(tempBuffer);
 for (int i = 0; i < n; i++) {
 if (off + i > b.length) {
 throw new ArrayIndexOutOfBoundsException(b.length);
 }
 b[off + i] = tempBuffer[i];
 }
 if (n == 0) {
 return -1;
 }
 return n;
 }

 /**
 * Reads the next byte of data from the activity audio stream if available.
 *
 * @return the next byte of data, or -1 if the end of the stream is reached
 * @see #read(byte[], int, int)
 * @see #read(byte[])
 * @see #available
 * <p>
 */
 @Override
 public int read() {
 byte[] data = new byte[1];
 int temp = read(data);
 if (temp <= 0) {
 // we have a weird situation if read(byte[]) returns 0!
 return -1;
 }
 return data[0] & 0xFF;
 }

 /**
 * Reads up to a specified maximum number of bytes of data from the activity audio stream,
 * putting them into the given byte array.
 *
 * @param b the buffer into which the data is read
 * @return the total number of bytes read into the buffer, or -1 if there
 * is no more data because the end of the stream has been reached
 */
 @Override
 public int read(byte[] b) {
 int n = (int) pullStreamImpl.read(b);
 if (n == 0) {
 return -1;
 }
 return n;
 }

 /**
 * Skips over and discards a specified number of bytes from this
 * audio input stream.
 *
 * @param n the requested number of bytes to be skipped
 * @return the actual number of bytes skipped
 * @throws IOException if an input or output error occurs
 * @see #read
 * @see #available
 */
 @Override
 public long skip(long n) {
 if (n <= 0) {
 return 0;
 }
 if (n <= Integer.MAX_VALUE) {
 byte[] tempBuffer = new byte[(int) n];
 return read(tempBuffer);
 }
 }
}

```

```

 long count = 0;
 for (long i = n; i > 0; i -= Integer.MAX_VALUE) {
 int size = (int) Math.min(Integer.MAX_VALUE, i);
 byte[] tempBuffer = new byte[size];
 count += read(tempBuffer);
 }
 return count;
 }

 /**
 * Closes this audio input stream and releases any system resources associated
 * with the stream.
 */
 @Override
 public void close() {
 this.pullStreamImpl.close();
 }

 /**
 * Fetch the audio format for the ActivityAudioStream. The ActivityAudioFormat defines the sample
 * rate, bits per sample, and the # channels.
 *
 * @return instance of the ActivityAudioFormat associated with the stream
 */
 public ActivityAudioStream.ActivityAudioFormat getActivityAudioFormat() {
 return activityAudioFormat;
 }

 /**
 * Returns the maximum number of bytes that can be read (or skipped over) from this
 * audio input stream without blocking.
 *
 * @return the number of bytes that can be read from this audio input stream without blocking.
 * As this implementation does not buffer, this will be defaulted to 0
 */
 @Override
 public int available() {
 return 0;
 }

 public ActivityAudioStream(final PullAudioOutputStream stream) {
 pullStreamImpl = stream;
 this.activityAudioFormat = new ActivityAudioStream.ActivityAudioFormat(SAMPLE_RATE,
BITS_PER_SECOND, CHANNELS, FRAME_SIZE, AudioEncoding.PCM_SIGNED);
 }

 private PullAudioOutputStream pullStreamImpl;

 private ActivityAudioFormat activityAudioFormat;

 /**
 * ActivityAudioFormat is an internal format which contains metadata regarding the type of
 * arrangement of
 * * audio bits in this activity audio stream.
 */
 static class ActivityAudioFormat {

 private long samplesPerSecond;
 private int bitsPerSample;
 private int channels;
 private int frameSize;
 private AudioEncoding encoding;

 public ActivityAudioFormat(long samplesPerSecond, int bitsPerSample, int channels, int
frameSize, AudioEncoding encoding) {
 this.samplesPerSecond = samplesPerSecond;
 this.bitsPerSample = bitsPerSample;
 this.channels = channels;
 this.encoding = encoding;
 }
 }
}

```

```

 this.frameSize = frameSize;
 }

 /**
 * Fetch the number of samples played per second for the associated audio stream format.
 *
 * @return the number of samples played per second
 */
 public long getSamplesPerSecond() {
 return samplesPerSecond;
 }

 /**
 * Fetch the number of bits in each sample of a sound that has this audio stream format.
 *
 * @return the number of bits per sample
 */
 public int getBitsPerSample() {
 return bitsPerSample;
 }

 /**
 * Fetch the number of audio channels used by this audio stream format.
 *
 * @return the number of channels
 */
 public int getChannels() {
 return channels;
 }

 /**
 * Fetch the default number of bytes in a frame required by this audio stream format.
 *
 * @return the number of bytes
 */
 public int getFrameSize() {
 return frameSize;
 }

 /**
 * Fetch the audio encoding type associated with this audio stream format.
 *
 * @return the encoding associated
 */
 public AudioEncoding getEncoding() {
 return encoding;
 }
}

/**
 * Enum defining the types of audio encoding supported by this stream.
 */
public enum AudioEncoding {
 PCM_SIGNED("PCM_SIGNED");

 String value;

 AudioEncoding(String value) {
 this.value = value;
 }
}

```

11. Save changes to the `ActivityAudioStream` file.

## Build and run the app

Select F11, or select **Run > Debug**. The console displays the message "Say something." At this point, speak an English phrase or sentence that your bot can understand. Your speech is transmitted to your bot through the Direct Line Speech channel where it's recognized and processed by your bot. The response is returned as an activity. If your bot returns speech as a response, the audio is played back by using the `AudioPlayer` class.

## Next steps

Additional samples, such as how to read speech from an audio file, are available on GitHub.

[Create and deploy a basic bot](#)

## See also

- [About voice assistants](#)
- [Get a Speech service subscription key for free](#)
- [Custom keywords](#)
- [Connect Direct Line Speech to your bot](#)
- [Explore Java samples on GitHub](#)

# Quickstart: Create a voice assistant in Java on Android by using the Speech SDK

12/10/2019 • 7 minutes to read • [Edit Online](#)

A quickstart is also available for [speech-to-text](#) and [text-to-speech](#).

In this article, you'll build a voice assistant with Java for Android using the [Speech SDK](#). This application will connect to a bot that you've already authored and configured with the [Direct Line Speech channel](#). It will then send a voice request to the bot and present a voice-enabled response activity.

This application is built with the Speech SDK Maven package and Android Studio 3.3. The Speech SDK is currently compatible with Android devices having 32/64-bit ARM and Intel x86/x64 compatible processors.

## NOTE

For the Speech Devices SDK and the Roobo device, see [Speech Devices SDK](#).

## Prerequisites

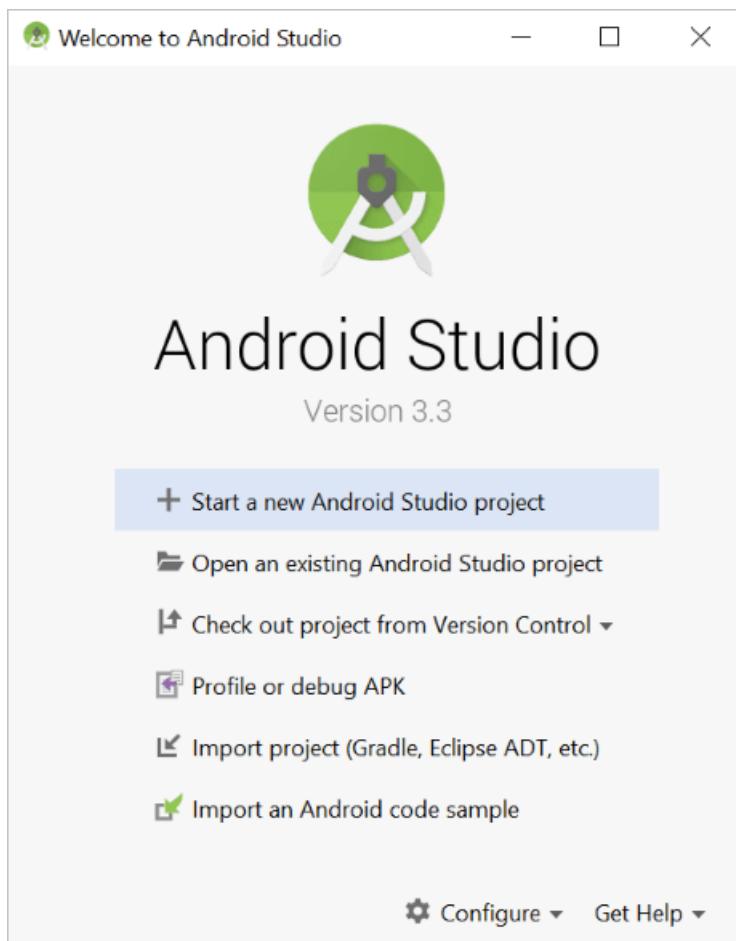
- An Azure subscription key for the Speech service. [Get one for free](#) or create it on the [Azure portal](#).
- A previously created bot configured with the [Direct Line Speech channel](#)
- [Android Studio v3.3 or later](#)

## NOTE

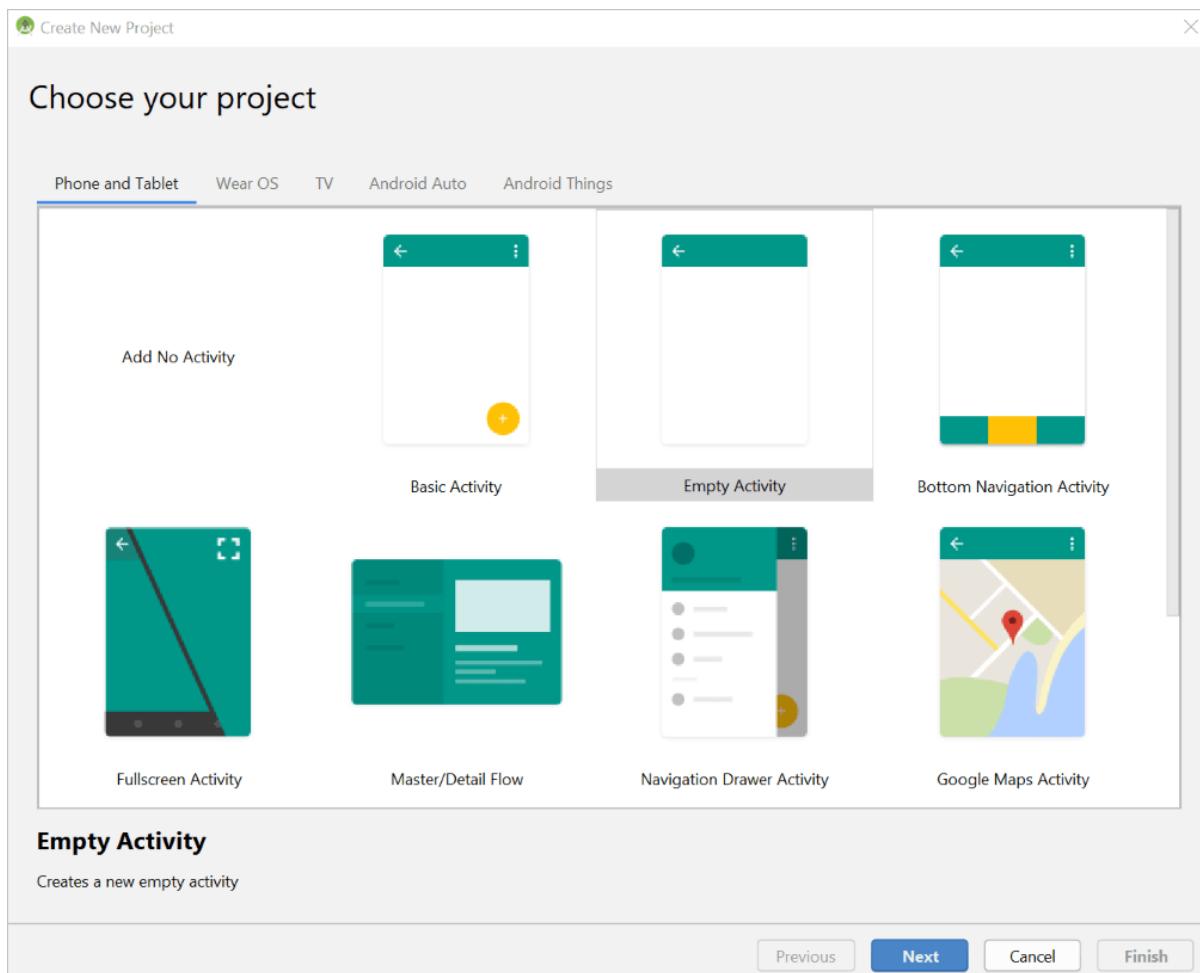
Please refer to the [list of supported regions for voice assistants](#) and ensure your resources are deployed in one of those regions.

## Create and configure a project

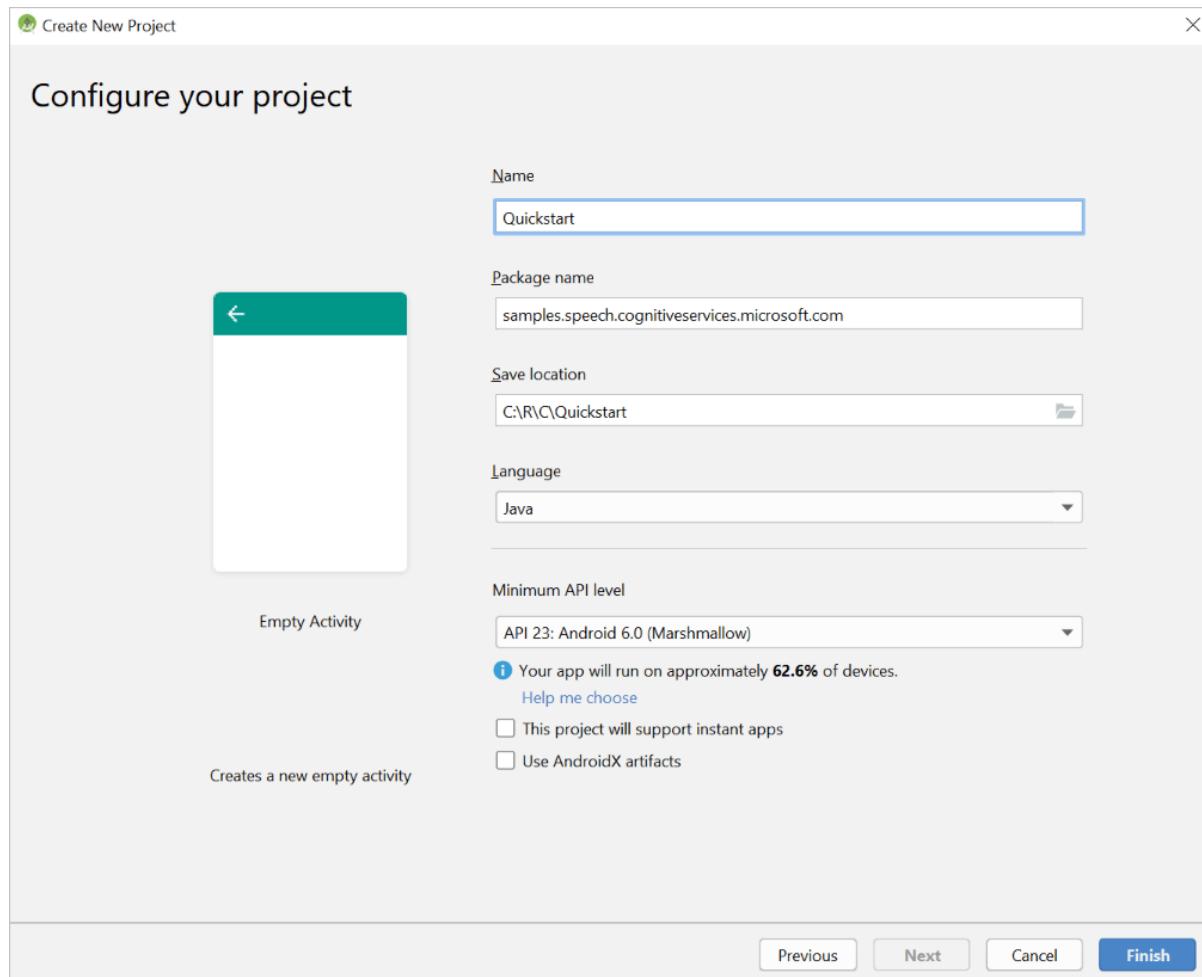
1. Launch Android Studio, and select **Start a new Android Studio project** in the **Welcome** window.



2. The **Choose your project** wizard appears. Select **Phone and Tablet** and **Empty Activity** in the activity selection box. Select **Next**.



3. On the **Configure your project** screen, enter *Quickstart* as **Name** and enter *samples.speech.cognitiveservices.microsoft.com* as **Package name**. Then select a project directory. For **Minimum API level**, select **API 23: Android 6.0 (Marshmallow)**. Leave all other check boxes clear, and select **Finish**.



Android Studio takes a moment to prepare your new Android project. Next, configure the project to know about the Azure Cognitive Services Speech SDK and to use Java 8.

**IMPORTANT**

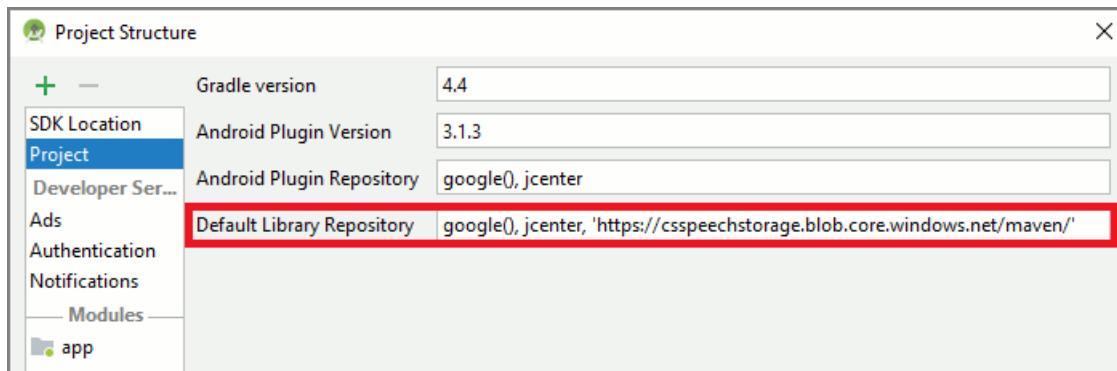
By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

The current version of the Cognitive Services Speech SDK is 1.7.0.

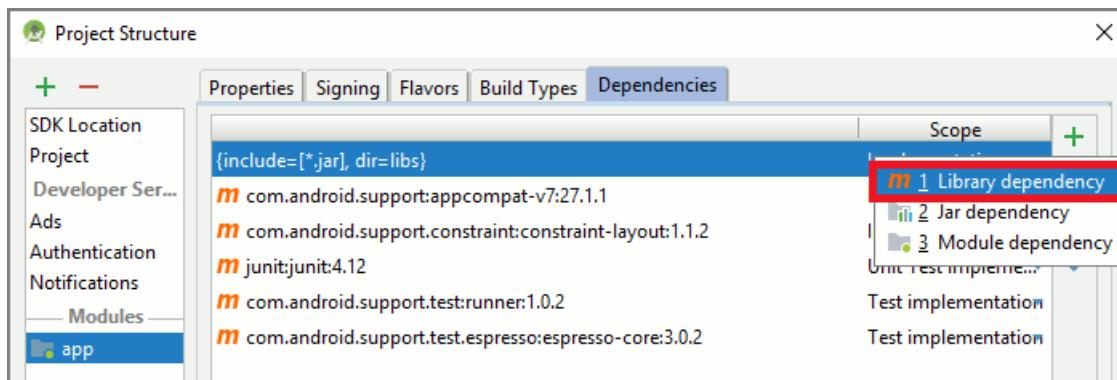
The Speech SDK for Android is packaged as an [AAR \(Android Library\)](#), which includes the necessary libraries and required Android permissions. It's hosted in a Maven repository at <https://csspeechstorage.blob.core.windows.net/maven/>.

Set up your project to use the Speech SDK. Open the **Project Structure** window by selecting **File > Project Structure** from the Android Studio menu bar. In the **Project Structure** window, make the following changes:

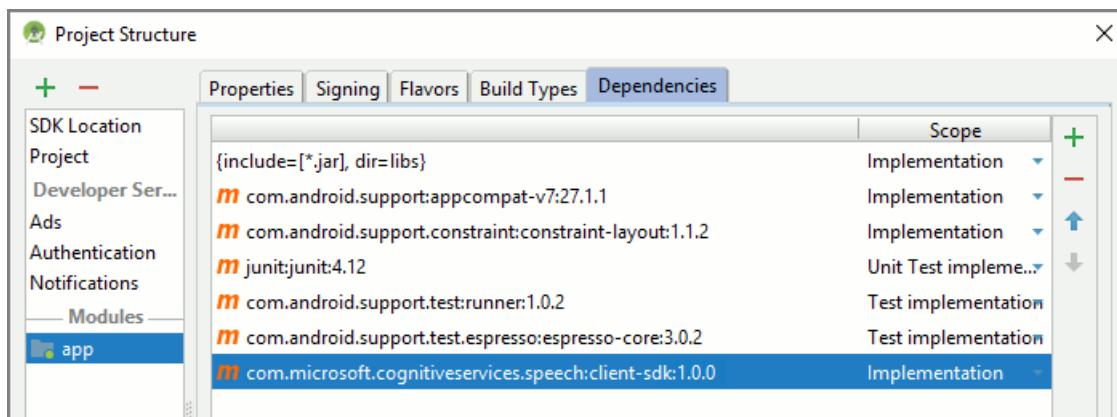
1. In the list on the left side of the window, select **Project**. Edit the **Default Library Repository** settings by appending a comma and our Maven repository URL enclosed in single quotation marks:  
`'https://csspeechstorage.blob.core.windows.net/maven/'`



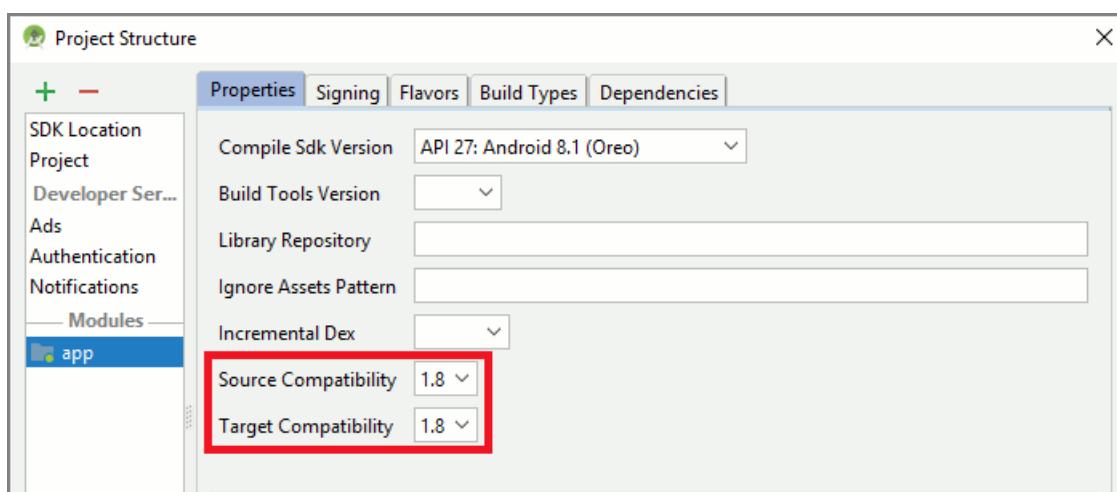
2. On the same screen, on the left side, select **app**. Then select the **Dependencies** tab at the top of the window. Select the green plus sign (+), and select **Library dependency** from the drop-down menu.



3. In the window that appears, enter the name and version of the Speech SDK for Android, `com.microsoft.cognitiveservices.speech:client-sdk:1.7.0`. Then select **OK**. The Speech SDK should be added to the list of dependencies now, as shown:



4. Select the **Properties** tab. For both **Source Compatibility** and **Target Compatibility**, select **1.8**.



5. Select **OK** to close the **Project Structure** window and apply your changes to the project.

## Create user interface

In this section, we'll create a basic user interface (UI) for the application. Let's start by opening the main activity: `activity_main.xml`. The basic template includes a title bar with the application's name, and a `TextView` with the message "Hello world!".

Next, replace the contents of the `activity_main.xml` with the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 tools:context=".MainActivity">

 <Button
 android:id="@+id/button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:onClick="onBotButtonClicked"
 android:text="Talk to your bot" />

 <TextView
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:text="Recognition Data"
 android:textSize="18dp"
 android:textStyle="bold" />

 <TextView
 android:id="@+id/recoText"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:text=" \n(Recognition goes here)\n" />

 <TextView
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:text="Activity Data"
 android:textSize="18dp"
 android:textStyle="bold" />

 <TextView
 android:id="@+id/activityText"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:scrollbars="vertical"
 android:text=" \n(Activities go here)\n" />

</LinearLayout>
```

This XML defines a simple UI to interact with your bot.

- The `button` element initiates an interaction and invokes the `onBotButtonClicked` method when clicked.
- The `recoText` element will display the speech-to-text results as you talk to your bot.
- The `activityText` element will display the JSON payload for the latest Bot Framework activity from your bot.

The text and graphical representation of your UI should now look like this:

TALK TO YOUR BOT

## Recognition Data

(Recognition goes here)

## Activity Data

(Activities go here)

## Add sample code

1. Open `MainActivity.java`, and replace the contents with the following code:

```
package samples.speech.cognitiveservices.microsoft.com;

import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioTrack;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
import android.view.View;
import android.widget.TextView;

import com.microsoft.cognitiveservices.speech.audio.AudioConfig;
import com.microsoft.cognitiveservices.speech.audio.PullAudioOutputStream;
import com.microsoft.cognitiveservices.speech.dialog.DialogServiceConfig;
import com.microsoft.cognitiveservices.speech.dialog.DialogServiceConnector;

import org.json.JSONException;
import org.json.JSONObject;

import static android.Manifest.permission.*;

public class MainActivity extends AppCompatActivity {
 // Replace below with your own speech subscription key
 private static String speechSubscriptionKey = "YourSpeechSubscriptionKey";
 // Replace below with your own speech service region
 private static String serviceRegion = "YourSpeechServiceRegion";

 private DialogServiceConnector connector;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 TextView recoText = (TextView) this.findViewById(R.id.recoText);
 TextView activityText = (TextView) this.findViewById(R.id.activityText);
 recoText.setMovementMethod(new ScrollingMovementMethod());
 activityText.setMovementMethod(new ScrollingMovementMethod());

 // Note: we need to request permissions for audio input and network access
 int requestCode = 5; // unique code for the permission request
 }
}
```

```

 ActivityCompat.requestPermissions(MainActivity.this, new String[]{RECORD_AUDIO, INTERNET},
requestCode);
 }

 public void onBotButtonClicked(View v) {
 // Recreate the DialogServiceConnector on each button press, ensuring that the existing one is
closed
 if (connector != null) {
 connector.close();
 connector = null;
 }

 // Create the DialogServiceConnector from speech subscription information
 BotFrameworkConfig config = BotFrameworkConfig.fromSubscription(speechSubscriptionKey,
serviceRegion);
 connector = new DialogServiceConnector(config, AudioConfig.fromDefaultMicrophoneInput());

 // Optional step: preemptively connect to reduce first interaction latency
 connector.connectAsync();

 // Register the DialogServiceConnector's event listeners
 registerEventListeners();

 // Begin sending audio to your bot
 connector.listenOnceAsync();
 }

 private void registerEventListeners() {
 TextView recoText = (TextView) this.findViewById(R.id.recoText); // 'recoText' is the ID of
your text view
 TextView activityText = (TextView) this.findViewById(R.id.activityText); // 'activityText' is
the ID of your text view

 // Recognizing will provide the intermediate recognized text while an audio stream is being
processed
 connector.recognizing.addEventListener((o, recoArgs) -> {
 recoText.setText(" Recognizing: " + recoArgs.getResult().getText());
 });

 // Recognized will provide the final recognized text once audio capture is completed
 connector.recognized.addEventListener((o, recoArgs) -> {
 recoText.setText(" Recognized: " + recoArgs.getResult().getText());
 });

 // SessionStarted will notify when audio begins flowing to the service for a turn
 connector.sessionStarted.addEventListener((o, sessionArgs) -> {
 recoText.setText("Listening...");
 });

 // SessionStopped will notify when a turn is complete and it's safe to begin listening again
 connector.sessionStopped.addEventListener((o, sessionArgs) -> {
 });

 // Canceled will be signaled when a turn is aborted or experiences an error condition
 connector.canceled.addEventListener((o, canceledArgs) -> {
 recoText.setText("Canceled (" + canceledArgs.getReason().toString() + ") error details: {" +
+ canceledArgs.getErrorDetails());
 connector.disconnectAsync();
 });

 // ActivityReceived is the main way your bot will communicate with the client and uses bot
framework activities.
 connector.activityReceived.addEventListener((o, activityArgs) -> {
 try {
 // Here we use JSONObject only to "pretty print" the condensed Activity JSON
 String rawActivity = activityArgs.getActivity().serialize();
 String formattedActivity = new JSONObject(rawActivity).toString(2);
 activityText.setText(formattedActivity);
 } catch (JSONException e) {

```

```
 activityText.setText("Couldn't format activity text: " + e.getMessage());
 }

 if (activityArgs.hasAudio()) {
 // Text-to-speech audio associated with the activity is 16 kHz 16-bit mono PCM data
 final int sampleRate = 16000;
 int bufferSize = AudioTrack.getMinBufferSize(sampleRate, AudioFormat.CHANNEL_OUT_MONO,
AudioFormat.ENCODING_PCM_16BIT);

 AudioTrack track = new AudioTrack(
 AudioManager.STREAM_MUSIC,
 sampleRate,
 AudioFormat.CHANNEL_OUT_MONO,
 AudioFormat.ENCODING_PCM_16BIT,
 bufferSize,
 AudioTrack.MODE_STREAM);

 track.play();

 PullAudioOutputStream stream = activityArgs.getAudio();

 // Audio is streamed as it becomes available. Play it as it arrives.
 byte[] buffer = new byte[bufferSize];
 long bytesRead = 0;

 do {
 bytesRead = stream.read(buffer);
 track.write(buffer, 0, (int) bytesRead);
 } while (bytesRead == bufferSize);

 track.release();
 }
});
```

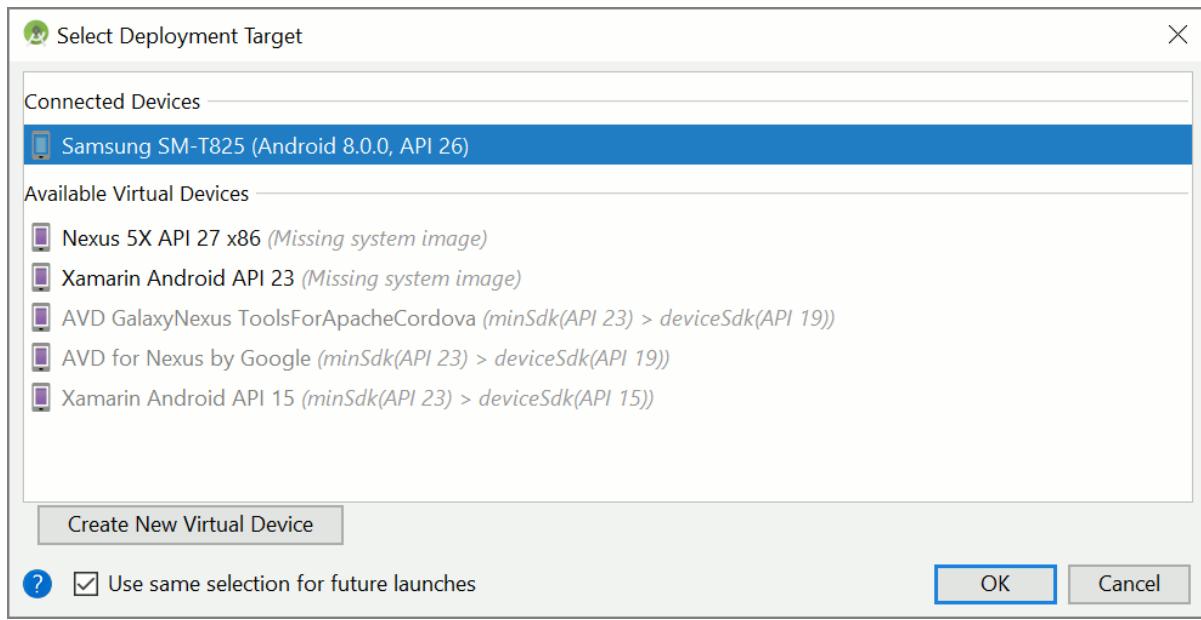
- The `onCreate` method includes code that requests microphone and internet permissions.
  - The method `onBotButtonClicked` is, as noted earlier, the button click handler. A button press triggers a single interaction ("turn") with your bot.
  - The `registerEventListeners` method demonstrates the events used by the `DialogServiceConnector` and basic handling of incoming activities.

2. In the same file, replace the configuration strings to match your resources:

- Replace `YourChannelSecret` with the Direct Line Speech channel secret for your bot.
  - Replace `YourSpeechSubscriptionKey` with your subscription key.
  - Replace `YourServiceRegion` with the [region](#) associated with your subscription. Only a subset of Speech service regions are currently supported with Direct Line Speech. For more information, see [regions](#).

## Build and run the app

1. Connect your Android device to your development PC. Make sure you have enabled [development mode](#) and [USB debugging](#) on the device.
  2. To build the application, press Ctrl+F9, or choose **Build > Make Project** from the menu bar.
  3. To launch the application, press Shift+F10, or choose **Run > Run 'app'**.
  4. In the deployment target window that appears, choose your Android device.



Once the application and its activity have launched, click the button to begin talking to your bot. Transcribed text will appear as you speak and the latest activity have you received from your bot will appear when it is received. If your bot is configured to provide spoken responses, the speech-to-text will automatically play.

## TALK TO YOUR BOT

### Recognition Data

Recognized: Hello there bot.

### Activity Data

```
{
 "channelData": {
 "conversationalAiData": {
 "requestInfo": {
 "interactionId": "d950b663-43e4-4ad7-a408-
4dc5d218b76f",
 "version": "0.2"
 }
 }
 },
 "channelId": "directlinespeech",
 "conversation": {
 "id": "dbae3b84-0717-4c2b-a29f-2d62b138209a",
 "isGroup": false
 },
 "from": {
 "id": "abigaildemo"
 },
 "id": "2814938a-fead-4aba-8fd7-4e025ebcfe7e",
 "inputHint": "acceptingInput",
 "recipient": {
 "id": "dbae3b84-0717-4c2b-a29f-2d62b138209a|0000"
 },
 "replyToId": "f95e8857-f7cc-4370-9fba-784c0d5b721e",
 "serviceUrl":
 "urn:botframework:websocket:directlinespeech",
 "speak": "<speak version=\"1.0\" xml:lang=\"en-us\"
xmlns=\"https://www.w3.org/2001/10/
synthesis\"><voice name=\"en-US-JessaNeural\"
xmlns=\"\">Hi!</voice></speak>",
 "text": "Hi!",
 ...
}
```



## Next steps

[Create and deploy a basic bot](#)

## See also

- [About voice assistants](#)
- [Get a Speech service subscription key for free](#)
- [Custom keywords](#)
- [Connect Direct Line Speech to your bot](#)
- [Explore Java samples on GitHub](#)

# Quickstart: Create a Custom Command (Preview)

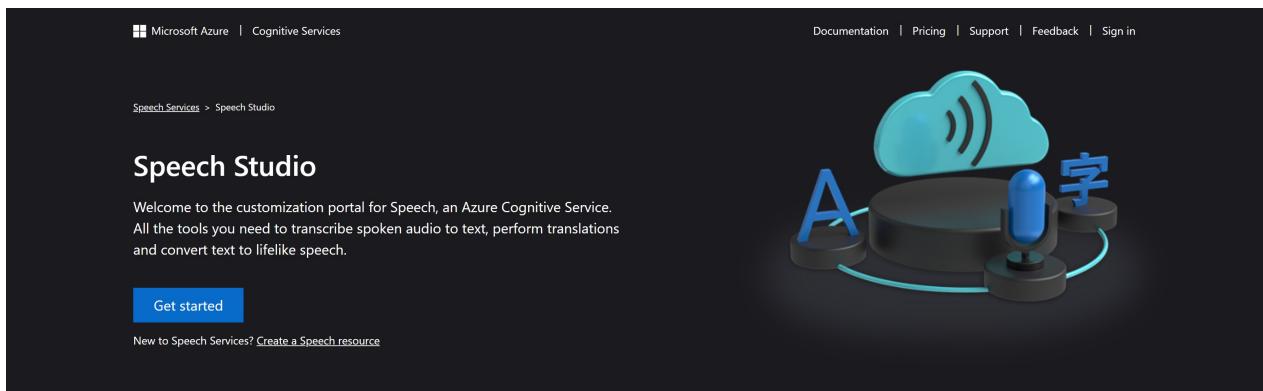
12/19/2019 • 3 minutes to read • [Edit Online](#)

In this article, you'll learn how to create and test a hosted Custom Commands application. The application will recognize an utterance like "turn on the tv" and respond with a simple message "Ok, turning on the tv".

## Prerequisites

- A Speech subscription.

If you don't have a speech subscription, you can create one by navigating to the [Speech Studio](#) and selecting **Create a Speech Resource**.



### NOTE

During preview, only the westus2 region is supported.

## Go to the Speech Studio for Custom Commands

1. Open your web browser, and navigate to the [Speech Studio](#)
2. Enter your credentials to sign in to the portal
  - The default view is your list of Speech subscriptions

### NOTE

If you don't see the select subscription page, you can navigate there by choosing "Speech resources" from the settings menu on the top bar.

3. Select your Speech subscription, then select **Go to Studio**
4. Select **Custom Commands (Preview)**

The default view is a list of the Custom Commands applications you created.

## Create a Custom Commands project

1. Select **New project** to create a new project

## New project

X

Name \*

Name your project

Description

Describe your project

Language \*

Select a language

Authoring Resource \*

Select a resource

[Create new resource](#)

[Cancel](#)

[Create](#)

2. Enter the project name and language.

3. Select an authoring resource. If there are no valid authoring resources, create one by selecting **Create new resource**.

## New LUIS Authoring Resource

X

Resource Name \*

Resource Group \*

Search for or create a new resource group

Location \*

Select a Location

Pricing Tier \*

Select a pricing tier

[Cancel](#)

[Create](#)

a. Enter the resource name, group, location, and pricing tier.

**NOTE**

You can create resource groups by entering the desired resource group name into the "Resource Group" field. The resource group will be created when **Create** is selected.

4. Click **Create** to create your project.

5. Once created, select your project.

Your view should now be an overview of your Custom Commands application.

## Update LUIS Resources (Optional)

You can update the authoring resource set in the new project window, and set a prediction resource used to recognize inputs during runtime.

**NOTE**

You will need to set a prediction resource before your application requests predictions beyond the 1,000 requests provided by the authoring resource.

The screenshot shows the 'Device Control Quickstart' page in Speech Studio. The left sidebar has 'Commands' and 'Settings' sections. The main area has a 'LUIS Resources' section with tabs for 'Custom Voice' and 'Remote Skills'. Below this is a detailed description of LUIS resources, fields for 'Authoring Resource' (set to 'resource1') and 'Prediction Resource' (set to 'Select a resource'), and a 'Create new resource' link. At the bottom right is a 'Save' button.

1. Navigate to the LUIS Resources pane by selecting **Settings** from the left pane, and then **LUIS Resources** from the middle pane.
2. Select a prediction resource, or create one by selecting **Create new resource**
3. Select **Save**

## Create a new Command

Now you can create a Command. Let's use an example that will take a single utterance, `turn on the tv`, and respond with the message `Ok, turning on the TV`.

1. Create a new Command by selecting the **+** icon next to commands and give it the name `TurnOn`
2. Select **Save**

Speech Studio &gt; Custom Commands

## Device Control Quickstart

English (United States)

Commands	+	Sample sentences
FallbackCommand		Parameters
<b>TurnOn</b>		Http Endpoints
Settings		Completion Rules
		Advanced Rules

A Command is a set of:

GROUP	DESCRIPTION
Sample Sentences	Example utterances the user can say to trigger this Command
Parameters	Information required to complete the Command
Completion Rules	The actions to be taken to fulfill the Command. For example, to respond to the user or communicate with another web service
Advanced Rules	Additional rules to handle more specific or complex situations

### Add a Sample Sentence

Let's start with Sample Sentences and provide an example of what the user can say:

```
turn on the tv
```

For now, we have no parameters so we can move on to Completion Rules.

### Add a Completion Rule

Now add a Completion Rule to respond to the user indicating that an action is being taken.

1. Create a new Completion Rule by selecting the icon next to Completion Rules
2. Enter the rule name
3. Add an action
  - a. Create a new Speech Response Action by selecting the icon next to Actions and select **SpeechResponse**
  - b. Enter the response

**NOTE**

Regular text must start with a dash. For more details, go [here](#)

## Edit Action

Type \*

SpeechResponse



Response template



1 - Ok, turning on the TV

4. Click **Save** to save the rule

## Completion Rules

Completion rules are executed once we're ready to fulfill, or complete, the command. They occur when all the required parameters are gathered.

### Rule Name ⓘ

Name \*

ConfirmationResponse

### Conditions ⓘ +

### Actions ⓘ +

SpeechResponse

- Ok, turning on the TV

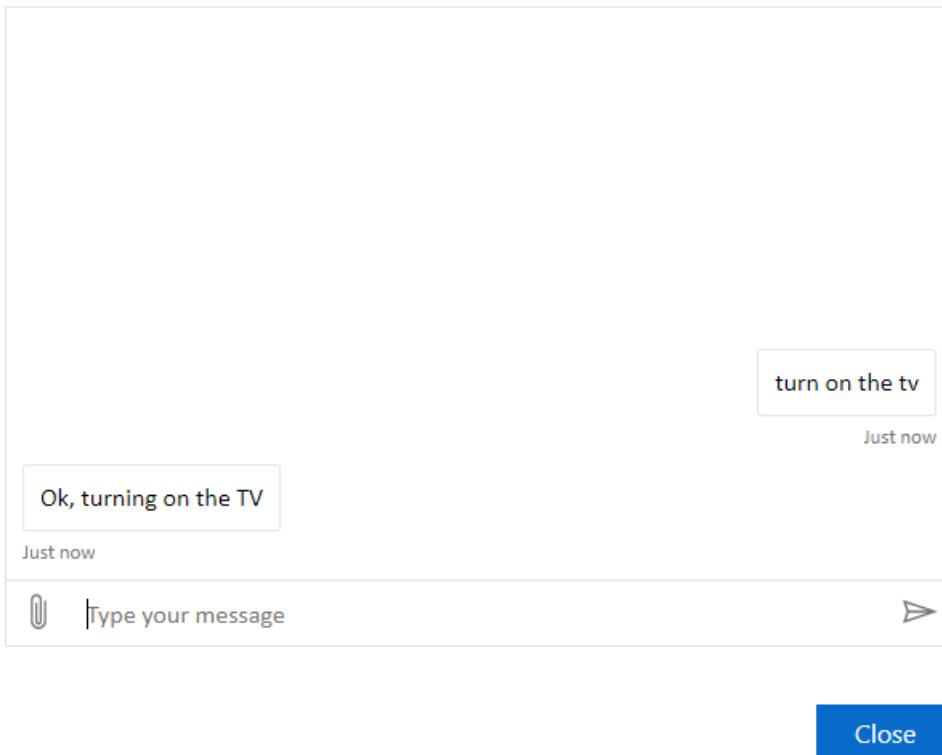
Save

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	"ConfirmationResponse"	A name describing the purpose of the rule
Conditions	None	Conditions that determine when the rule can run
Actions	SpeechResponse "- Ok, turning on the TV"	The action to take when the rule condition is true

## Try it out

Test the behavior using the Test chat panel.

Test your application



- You type: "turn on the tv"
- Expected response: "Ok, turning on the tv"

## Next steps

[Quickstart: Create a Custom Command with Parameters \(Preview\)](#)

# Quickstart: Create a Custom Command with parameters (Preview)

12/10/2019 • 3 minutes to read • [Edit Online](#)

In the [previous article](#), we created a new Custom Commands project to respond to commands without parameters.

In this article, we will extend this application with parameters so that it can handle turning on and turning off multiple devices.

## Create Parameters

1. Open the project [we created previously](#)
2. Because the Command will now handle on and off, rename the Command to "TurnOnOff"
  - Hover over the name of the Command and select the edit icon to change the name
3. Create a new parameter to represent whether the user wants to turn the device on or off
  - Select the  icon next to the Parameters section

# Parameters

A parameter is a variable that's filled during dialogue with a Custom Commands application. [Learn more](#)

## Parameter details

Name \*

OnOff

Is Global

Required

Response template \*

1 - On or off?

Type \*

String

Default Value

Configuration \*

String List

## String list values



on

off

Save

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	OnOff	A descriptive name for your parameter

SETTING	SUGGESTED VALUE	DESCRIPTION
Is Global	unchecked	Checkbox indicating whether a value for this parameter is globally applied to all Commands in the project
Required	checked	Checkbox indicating whether a value for this parameter is required before completing the Command
Response template	"- On or off?"	A prompt to ask for the value of this parameter when it isn't known
Type	String	The type of parameter, such as Number, String, or Date Time
Configuration	String List	For Strings, a String List limits inputs to a set of possible values
String list values	on, off	For a String List parameter, the set of possible values and their synonyms

- Next, select the  icon again to add a second parameter to represent the name of the devices. For this example, a tv and a fan

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	SubjectDevice	A descriptive name for your parameter
Is Global	unchecked	Checkbox indicating whether a value for this parameter is globally applied to all Commands in the project
Required	checked	Checkbox indicating whether a value for this parameter is required before completing the Command
Response template	"- Which device?"	A prompt to ask for the value of this parameter when it isn't known
Type	String	The type of parameter, such as Number, String, or Date Time
Configuration	String List	For Strings, a String List limits inputs to a set of possible values
String list values	tv, fan	For a String List parameter, the set of possible values and their synonyms
Synonyms (tv)	television, telly	Optional synonyms for each possible value of a String List Parameter

## Add Sample Sentences

With parameters, it's helpful to add sample sentences that cover all possible combinations. For example:

1. Full parameter information - "turn {OnOff} the {SubjectDevice}"
2. Partial parameter information - "turn it {OnOff}"
3. No parameter information - "turn something"

Sample sentences with different amounts of information allow the Custom Commands application to resolve both one-shot resolutions and multi-turn resolutions with partial information.

With that in mind, edit the Sample Sentences to use the parameters as suggested below.

#### TIP

In the Sample Sentences editor use curly braces to refer to your parameters. - turn {OnOff} the {SubjectDevice} Use tab completion to refer to previously created parameters.

```

1 turn {OnOff} the {SubjectDevice}
2 {SubjectDevice} {OnOff}
3 turn it {OnOff}
4 turn something {OnOff}
5 turn something

```

```

turn {OnOff} the {SubjectDevice}
{SubjectDevice} {OnOff}
turn it {OnOff}
turn something {OnOff}
turn something

```

## Add parameters to Completion rule

Modify the Completion rule that you created in [the previous quickstart](#):

1. Add a new Condition and select Required parameter. Select both `onoff` and `SubjectDevice`
2. Edit the Speech Response action to use `onoff` and `SubjectDevice`:

```
Ok, turning {OnOff} the {SubjectDevice}
```

## Try it out

Open the Test chat panel and try a few interactions.

- Input: turn off the tv
- Output: Ok, turning off the tv
- Input: turn off the television
- Output: Ok, turning off the tv
- Input: turn it off
- Output: Which device?
- Input: the tv
- Output: Ok, turning off the tv

## Next steps

[Quickstart: Connect to a Custom Command application with the Speech SDK \(Preview\)](#)

# Quickstart: Use Custom Commands with Custom Voice (Preview)

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

In the [previous article](#), we created a new Custom Commands project to respond to commands with parameters.

In this article, we'll select a custom output voice for the application we created.

## Select a Custom Voice

1. Open the project [we created previously](#)
2. Select **Settings** from the left pane
3. Select **Custom Voice** from the middle pane
4. Select the desired custom or public voice from the table
5. Select **Save**

The screenshot shows the Microsoft Speech Studio interface. In the top navigation bar, it says "Speech Studio > Custom Commands" and "Device Control Quickstart English (United States)". On the right, there are buttons for "Train", "Test", and "Publish". The left sidebar has sections for "Commands" (with items like "FallbackCommand", "TurnOnOff", "SetTemperature") and "Settings". The main area is titled "Custom Voice" and contains a sub-section "Custom Voice". It says, "On this page, select your desired output voice for interactions with this project. You can use a pre-built voice model, or build your own Custom Voice model." Below this is a table with columns "Name", "Gender", and "Type". The table lists the following voices:

Name	Gender	Type
GuyNeural	Male	Neural
JessaNeural	Female	Neural
BenjaminRUS	Male	Standard
Guy24kRUS	Male	Standard
Jessa24kRUS	Female	Standard
JessaRUS	Female	Standard

A "Save" button is located at the bottom right of the main area.

### NOTE

Custom voices can be created from the Custom Voice project page. Select the **Speech Studio** link, then **Custom Voice** to get started.

Now the application will respond in the selected voice, instead of the default voice.

## Next steps

[Quickstart: Connect to a Custom Command application with the Speech SDK \(Preview\)](#)

# Quickstart: Connect to a Custom Commands application with the Speech SDK (Preview)

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

After creating a hosted Custom Commands application, you can begin talking to it from a client device.

In this article, you'll:

- Publish a Custom Commands application and get an application identifier (App ID)
- Create a client app using the Speech SDK to allow you to talk to your Custom Commands application

## Prerequisites

A Custom Commands application is required to complete this article. If you haven't created a Custom Commands application yet, you can do so in these previous quickstarts:

- [Quickstart: Create a Custom Command \(Preview\)](#)
- [Quickstart: Create a Custom Command with Parameters \(Preview\)](#)

You'll also need:

- [Visual Studio 2019](#)
- An Azure subscription key for Speech Services. [Get one for free](#) or create it on the [Azure portal](#)

## Optional: Get started fast

This quickstart describes, step by step, how to make a client application to connect to your Custom Commands app. If you prefer to dive right in, the complete, ready-to-compile source code used in this quickstart is available in the [Speech SDK Samples](#) under the `quickstart` folder.

## Step 1: Publish Custom Commands application

1. Open your [previously created Custom Commands application](#) and select **Publish**

### Publish your application

**Device Control Quickstart**'s has been successfully trained and published.

To use your application from the Speech SDK.

- Enter the speech command app id [REDACTED] in your client application.
- Enter one of your subscription keys in your client application.
- [Learn more.](#)

2. Copy the App ID from the publish notification for later use

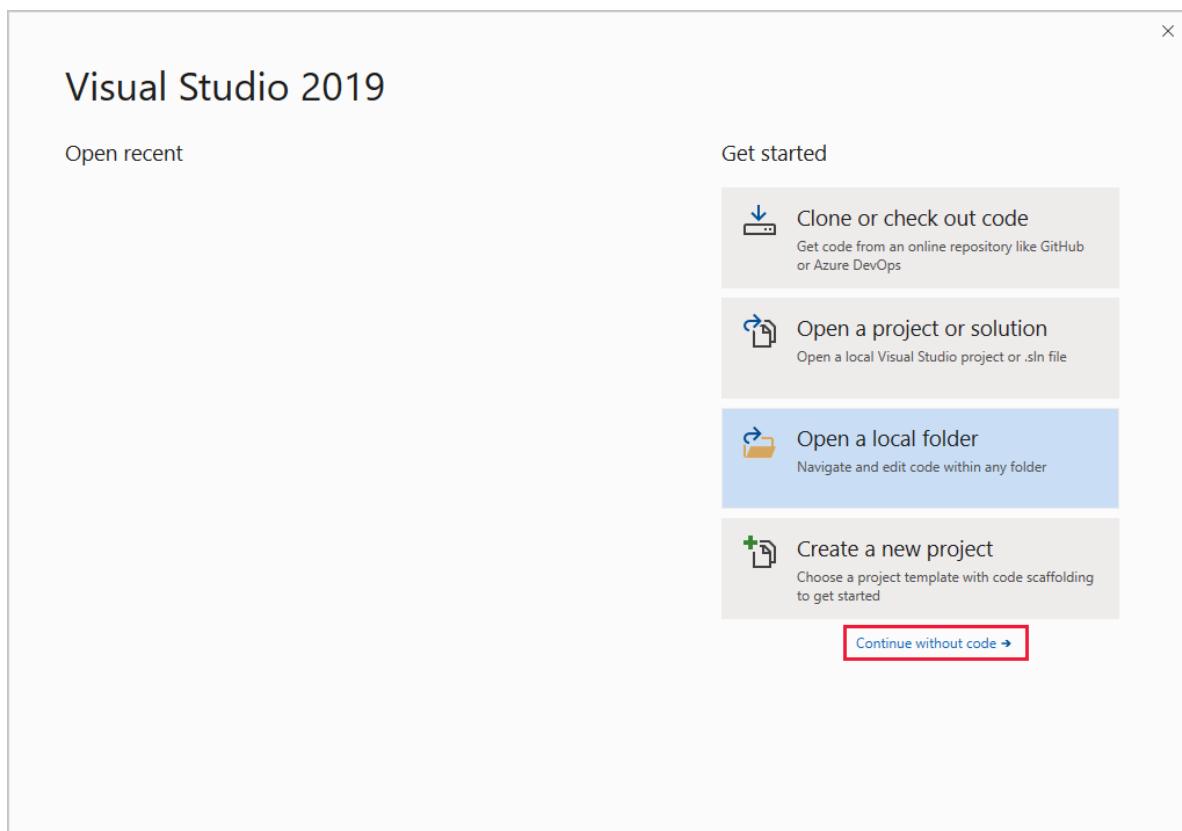
## Step 2: Create a Visual Studio project

To create a Visual Studio project for Universal Windows Platform (UWP) development, you need to set up Visual Studio development options, create the project, select the target architecture, set up audio capture, and install the Speech SDK.

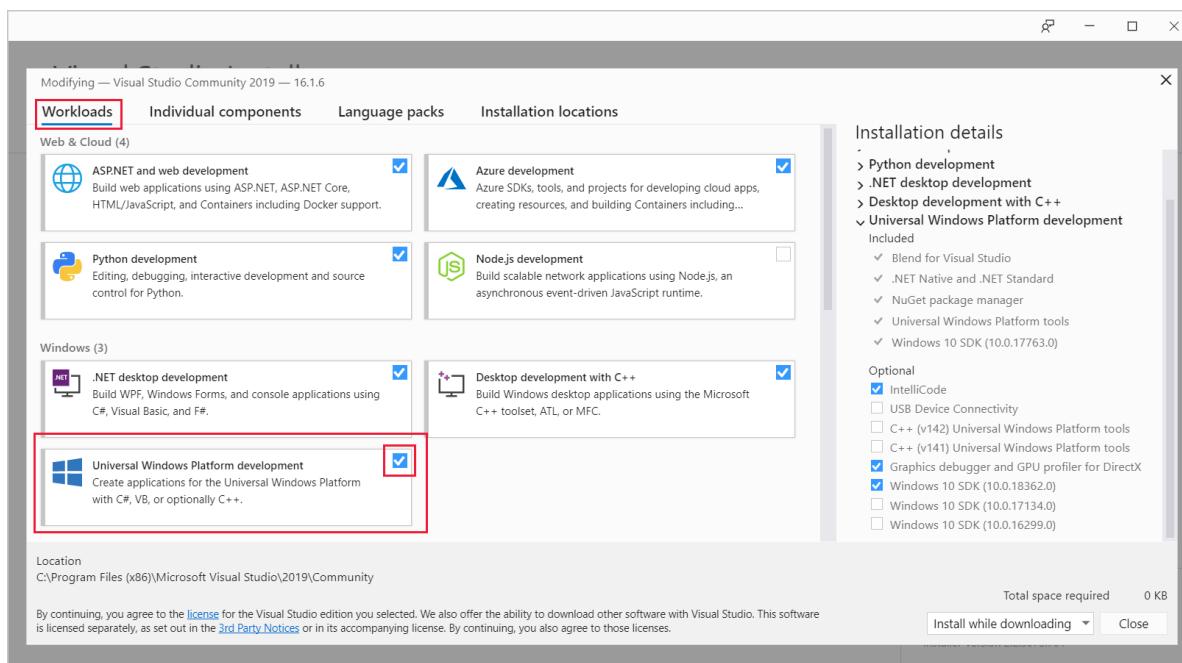
## Set up Visual Studio development options

To start, make sure you're set up correctly in Visual Studio for UWP development:

1. Open Visual Studio 2019 to display the **Start** window.



2. Select **Continue without code** to go to the Visual Studio IDE.
3. From the Visual Studio menu bar, select **Tools > Get Tools and Features** to open Visual Studio Installer and view the **Modifying** dialog box.



4. In the **Workloads** tab, under **Windows**, find the **Universal Windows Platform development** workload. If the check box next to that workload is already selected, close the **Modifying** dialog box, and go to step 6.
5. Select the **Universal Windows Platform development** check box, select **Modify**, and then in the **Before we get started** dialog box, select **Continue** to install the UWP development workload. Installation of the

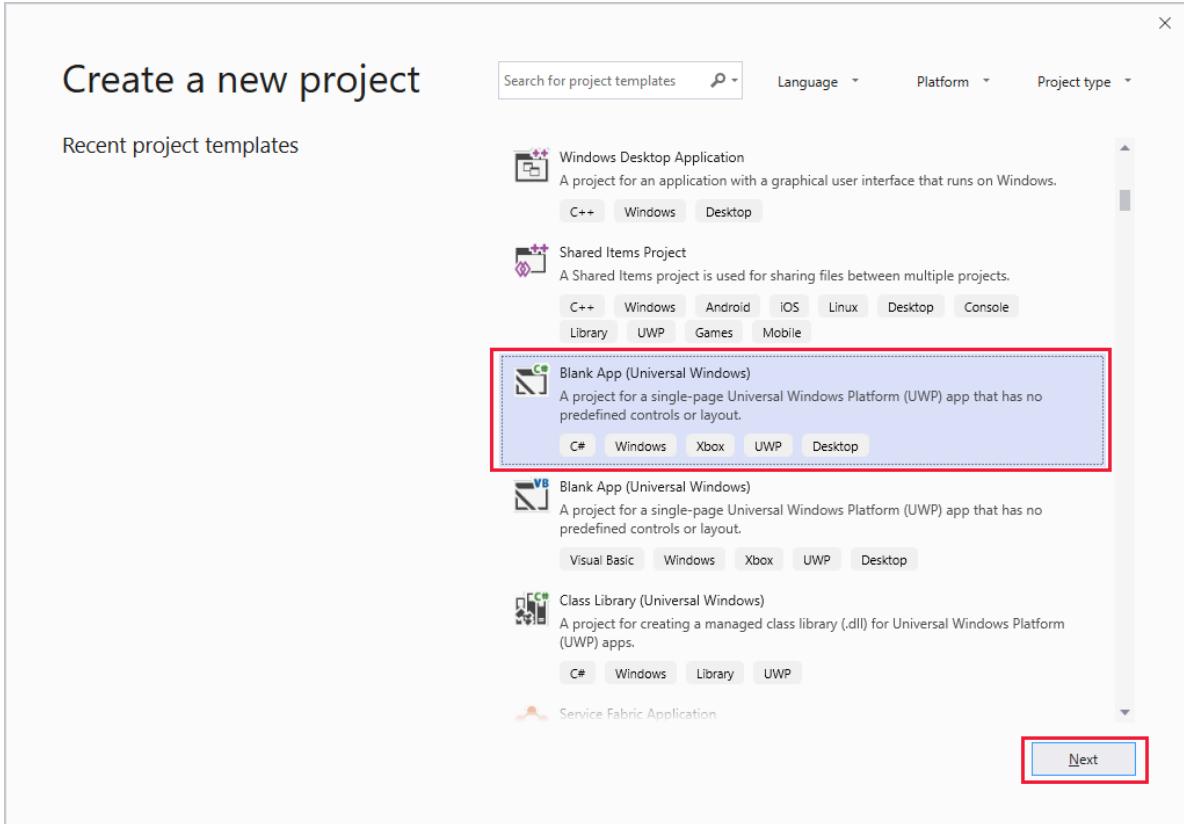
new feature may take a while.

6. Close Visual Studio Installer.

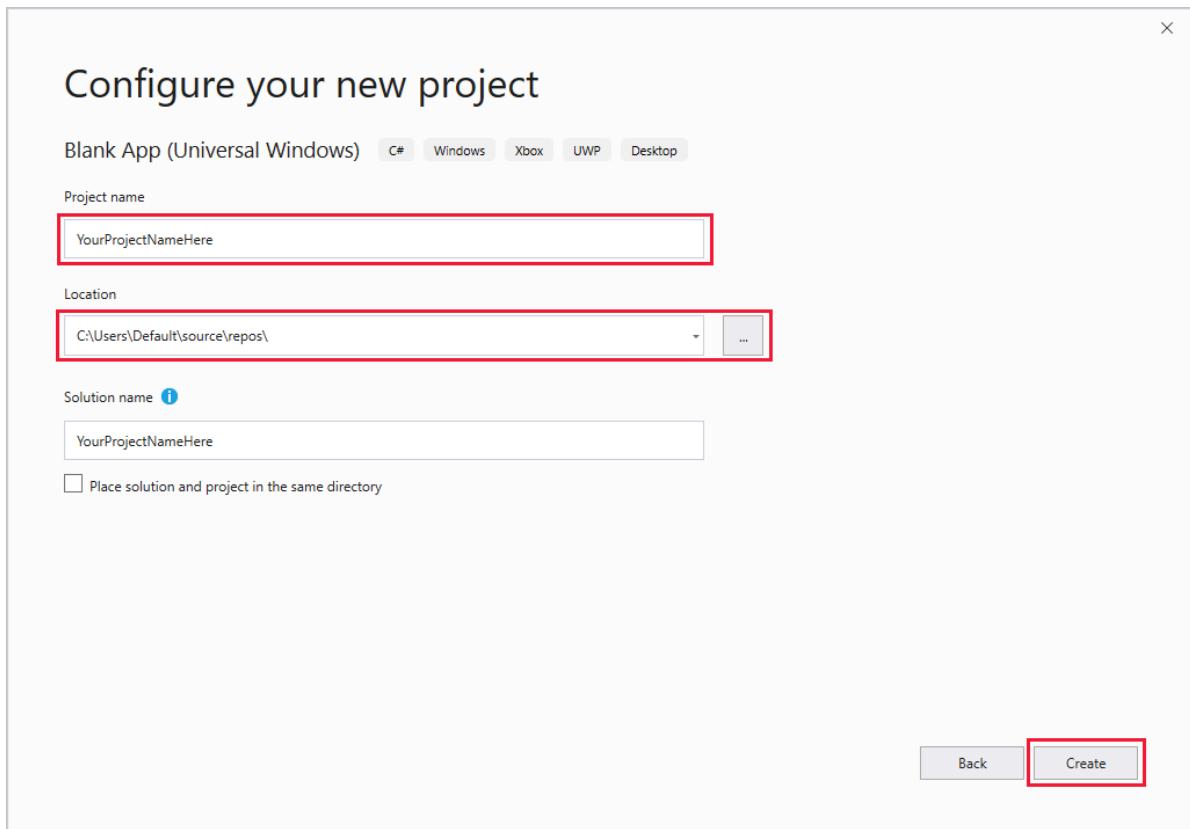
### Create the project and select the target architecture

Next, create your project:

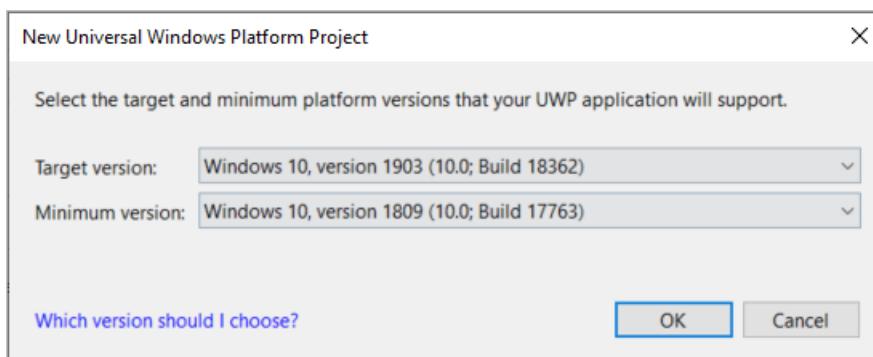
1. In the Visual Studio menu bar, choose **File > New > Project** to display the **Create a new project** window.



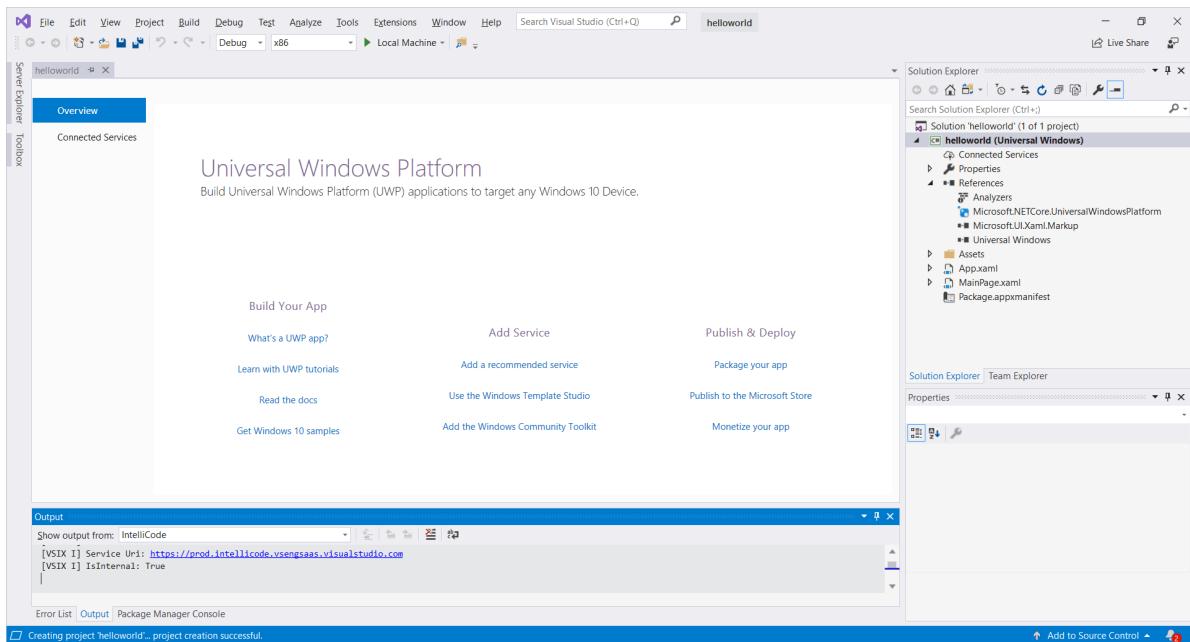
2. Find and select **Blank App (Universal Windows)**. Make sure that you select the C# version of this project type (as opposed to Visual Basic).
3. Select **Next** to display the **Configure your new project** screen.



4. In **Project name**, enter `helloworld`.
5. In **Location**, navigate to and select or create the folder to save your project in.
6. Select **Create** to go to the **New Universal Windows Platform Project** window.



7. In **Minimum version** (the second drop-down box), choose **Windows 10 Fall Creators Update (10.0; Build 16299)**, which is the minimum requirement for the Speech SDK.
8. In **Target version** (the first drop-down box), choose a value identical to or later than the value in **Minimum version**.
9. Select **OK**. You're returned to the Visual Studio IDE, with the new project created and visible in the **Solution Explorer** pane.



Now select your target platform architecture. In the Visual Studio toolbar, find the **Solution Platforms** drop-down box. (If you don't see it, choose **View > Toolbars > Standard** to display the toolbar containing **Solution Platforms**.) If you're running 64-bit Windows, choose **x64** in the drop-down box. 64-bit Windows can also run 32-bit applications, so you can choose **x86** if you prefer.

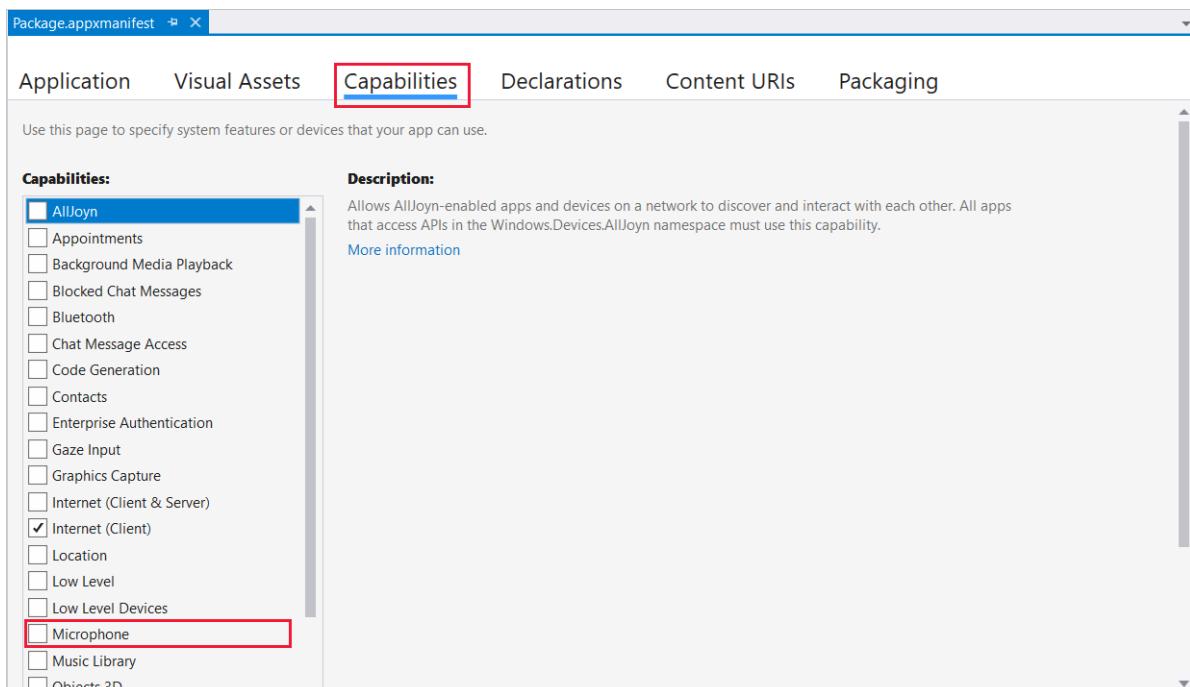
#### NOTE

The Speech SDK only supports Intel-compatible processors. ARM processors are currently not supported.

### Set up audio capture

Then allow the project to capture audio input:

1. In **Solution Explorer**, double-click **Package.appxmanifest** to open the package application manifest.
2. Select the **Capabilities** tab.

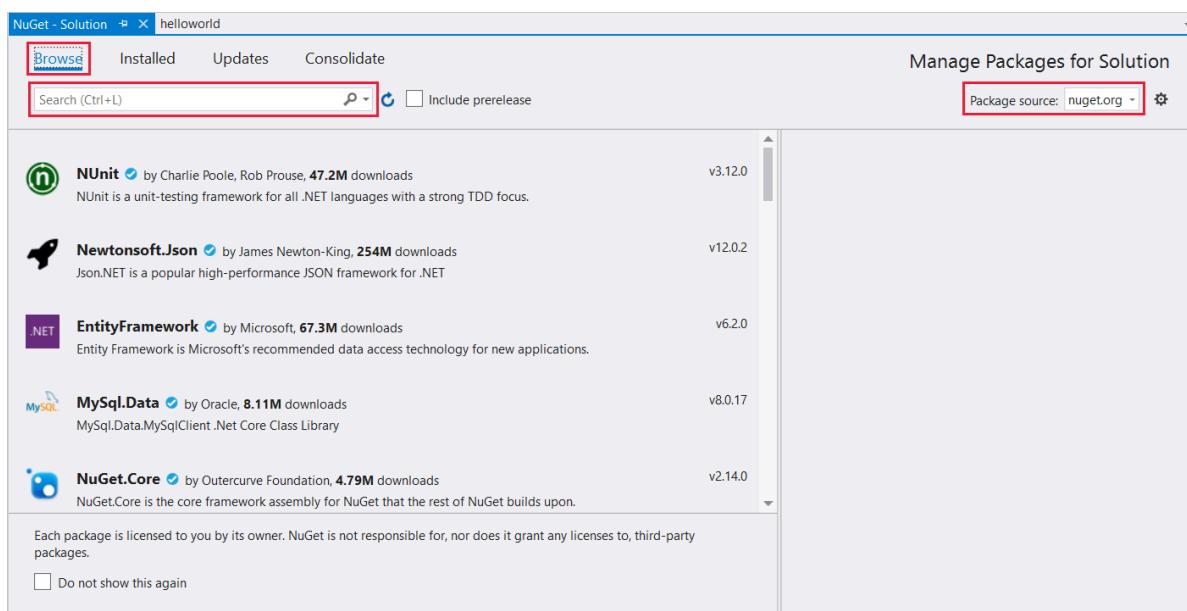


3. Select the box for the **Microphone** capability.
4. From the menu bar, choose **File > Save Package.appxmanifest** to save your changes.

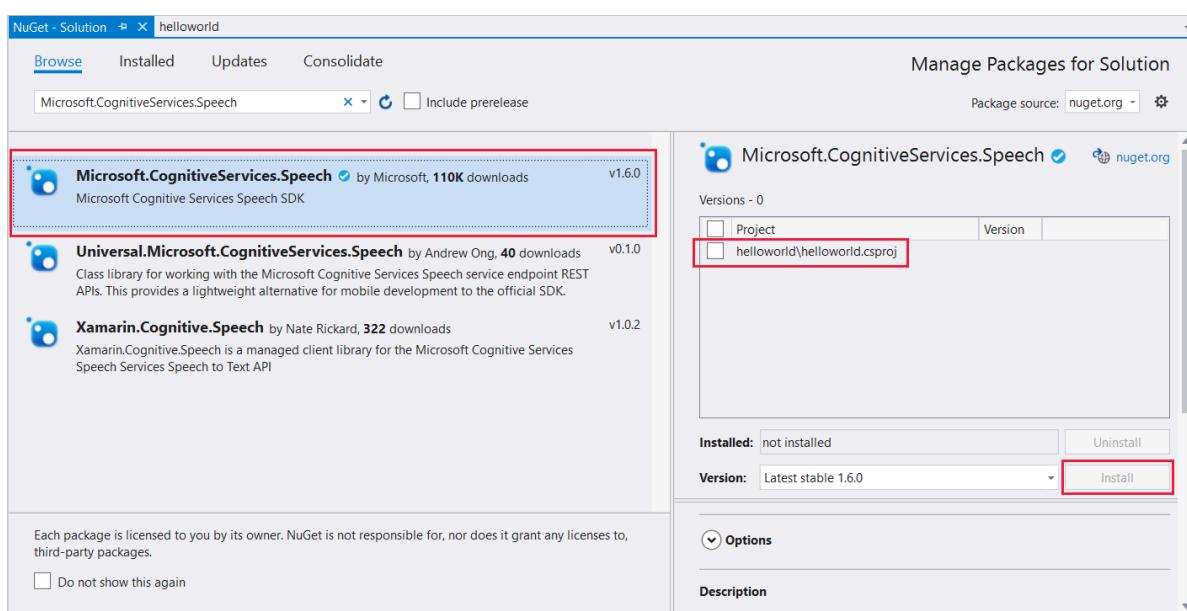
## Install the Speech SDK

Finally, install the [Speech SDK NuGet package](#), and reference the Speech SDK in your project:

1. In **Solution Explorer**, right-click your solution, and choose **Manage NuGet Packages for Solution** to go to the **NuGet - Solution** window.
2. Select **Browse**.



3. In **Package source**, choose **nuget.org**.
4. In the **Search** box, enter **Microsoft.CognitiveServices.Speech**, and then choose that package after it appears in the search results.



5. In the package status pane next to the search results, select your **helloworld** project.
6. Select **Install**.
7. In the **Preview Changes** dialog box, select **OK**.
8. In the **License Acceptance** dialog box, view the license, and then select **I Accept**. The package installation begins, and when installation is complete, the **Output** pane displays a message similar to the following text:  
Successfully installed 'Microsoft.CognitiveServices.Speech 1.7.0' to helloworld.

## Step 3: Add sample code

In this step we add the XAML code that defines the user interface of the application, and add the C# code-behind implementation.

### XAML code

Create the application's user interface by adding the XAML code.

1. In **Solution Explorer**, open `MainPage.xaml`
2. In the designer's XAML view, replace the entire contents with the following code snippet:

```
<Page
 x:Class="helloworld.MainPage"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:local="using:helloworld"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 mc:Ignorable="d"
 Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

 <Grid>
 <StackPanel Orientation="Vertical" HorizontalAlignment="Center"
 Margin="20,50,0,0" VerticalAlignment="Center" Width="800">
 <Button x:Name="EnableMicrophoneButton" Content="Enable Microphone"
 Margin="0,0,10,0" Click="EnableMicrophone_ButtonClicked"
 Height="35"/>
 <Button x:Name="ListenButton" Content="Talk"
 Margin="0,10,10,0" Click="ListenButton_ButtonClicked"
 Height="35"/>
 <StackPanel x:Name="StatusLabel" Orientation="Vertical"
 RelativePanel.AlignBottomWithPanel="True"
 RelativePanel.AlignRightWithPanel="True"
 RelativePanel.AlignLeftWithPanel="True">
 <TextBlock x:Name="StatusLabel" Margin="0,10,10,0"
 TextWrapping="Wrap" Text="Status:" FontSize="20"/>
 <Border x:Name="StatusBorder" Margin="0,0,0,0">
 <ScrollViewer VerticalScrollMode="Auto"
 VerticalScrollBarVisibility="Auto" MaxHeight="200">
 <!-- Use LiveSetting to enable screen readers to announce
 the status update. -->
 <TextBlock
 x:Name="StatusBlock" FontWeight="Bold"
 AutomationProperties.LiveSetting="Assertive"
 MaxWidth="{Binding ElementName=Splitter, Path=ActualWidth}"
 Margin="10,10,10,20" TextWrapping="Wrap" />
 </ScrollViewer>
 </Border>
 </StackPanel>
 <MediaElement x:Name="mediaElement"/>
 </Grid>
</Page>
```

The Design view is updated to show the application's user interface.

### C# code-behind source

Add the code-behind source so that the application works as expected. The code-behind source includes:

- Required `using` statements for the `Speech` and `Speech.Dialog` namespaces
- A simple implementation to ensure microphone access, wired to a button handler
- Basic UI helpers to present messages and errors in the application

- A landing point for the initialization code path that will be populated later
- A helper to play back text-to-speech (without streaming support)
- An empty button handler to start listening that will be populated later

Add the code-behind source as follows:

1. In **Solution Explorer**, open the code-behind source file `MainPage.xaml.cs` (grouped under `MainPage.xaml`)
2. Replace the file's contents with the following code:

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Dialog;
using System;
using System.Diagnostics;
using System.IO;
using System.Text;
using Windows.Foundation;
using Windows.Storage.Streams;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;

namespace helloworld
{
 public sealed partial class MainPage : Page
 {
 private DialogServiceConnector connector;

 private enum NotifyType
 {
 StatusMessage,
 ErrorMessage
 };

 public MainPage()
 {
 this.InitializeComponent();
 }

 private async void EnableMicrophone_ButtonClicked(
 object sender, RoutedEventArgs e)
 {
 bool isMicAvailable = true;
 try
 {
 var mediaCapture = new Windows.Media.Capture.MediaCapture();
 var settings =
 new Windows.Media.Capture.MediaCaptureInitializationSettings();
 settings.StreamingCaptureMode =
 Windows.Media.Capture.StreamingCaptureMode.Audio;
 await mediaCapture.InitializeAsync(settings);
 }
 catch (Exception)
 {
 isMicAvailable = false;
 }
 if (!isMicAvailable)
 {
 await Windows.System.Launcher.LaunchUriAsync(
 new Uri("ms-settings:privacy-microphone"));
 }
 else
 {
 NotifyUser("Microphone was enabled", NotifyType.StatusMessage);
 }
 }
 }
}
```

```

}

private void NotifyUser(
 string strMessage, NotifyType type = NotifyType.StatusMessage)
{
 // If called from the UI thread, then update immediately.
 // Otherwise, schedule a task on the UI thread to perform the update.
 if (Dispatcher.HasThreadAccess)
 {
 UpdateStatus(strMessage, type);
 }
 else
 {
 var task = Dispatcher.RunAsync(
 Windows.UI.Core.CoreDispatcherPriority.Normal,
 () => UpdateStatus(strMessage, type));
 }
}

private void UpdateStatus(string strMessage, NotifyType type)
{
 switch (type)
 {
 case NotifyType.StatusMessage:
 StatusBorder.Background = new SolidColorBrush(
 Windows.UI.Colors.Green);
 break;
 case NotifyType.ErrorMessage:
 StatusBorder.Background = new SolidColorBrush(
 Windows.UI.Colors.Red);
 break;
 }
 StatusBlock.Text += string.IsNullOrEmpty(StatusBlock.Text)
 ? strMessage : "\n" + strMessage;

 if (!string.IsNullOrEmpty(StatusBlock.Text))
 {
 StatusBorder.Visibility = Visibility.Visible;
 StatusPanel.Visibility = Visibility.Visible;
 }
 else
 {
 StatusBorder.Visibility = Visibility.Collapsed;
 StatusPanel.Visibility = Visibility.Collapsed;
 }
 // Raise an event if necessary to enable a screen reader
 // to announce the status update.
 var peer =
 Windows.UI.Xaml.Automation.Peers.FrameworkElementAutomationPeer.FromElement(StatusBlock);
 if (peer != null)
 {
 peer.RaiseAutomationEvent(
 Windows.UI.Xaml.Automation.Peers.AutomationEvents.LiveRegionChanged);
 }
}

// Waits for and accumulates all audio associated with a given
// PullAudioOutputStream and then plays it to the MediaElement. Long spoken
// audio will create extra latency and a streaming playback solution
// (that plays audio while it continues to be received) should be used --
// see the samples for examples of this.
private void SynchronouslyPlayActivityAudio(
 PullAudioOutputStream activityAudio)
{
 var playbackStreamWithHeader = new MemoryStream();
 playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("RIFF"), 0, 4); // ChunkID
 playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); // ChunkSize:
 max
 playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("WAVE"), 0, 4); // Format
}

```

```

playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("fmt "), 0, 4); // Subchunk1ID
playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 4); // Subchunk1Size: PCM
playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // AudioFormat: PCM
playbackStreamWithHeader.Write(BitConverter.GetBytes(1), 0, 2); // NumChannels: mono
playbackStreamWithHeader.Write(BitConverter.GetBytes(16000), 0, 4); // SampleRate: 16kHz
playbackStreamWithHeader.Write(BitConverter.GetBytes(32000), 0, 4); // ByteRate
playbackStreamWithHeader.Write(BitConverter.GetBytes(2), 0, 2); // BlockAlign
playbackStreamWithHeader.Write(BitConverter.GetBytes(16), 0, 2); // BitsPerSample: 16-bit
playbackStreamWithHeader.Write(Encoding.ASCII.GetBytes("data"), 0, 4); // Subchunk2ID
playbackStreamWithHeader.Write(BitConverter.GetBytes(UInt32.MaxValue), 0, 4); //

Subchunk2Size

byte[] pullBuffer = new byte[2056];

uint lastRead = 0;
do
{
 lastRead = activityAudio.Read(pullBuffer);
 playbackStreamWithHeader.Write(pullBuffer, 0, (int)lastRead);
}
while (lastRead == pullBuffer.Length);

var task = Dispatcher.RunAsync(
 Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
{
 mediaElement.SetSource(
 playbackStreamWithHeader.AsRandomAccessStream(), "audio/wav");
 mediaElement.Play();
});
}

private void InitializeDialogServiceConnector()
{
 // New code will go here
}

private async void ListenButton_Clicked(
 object sender, RoutedEventArgs e)
{
 // New code will go here
}
}
}

```

3. Add the following code to the method body of `InitializeDialogServiceConnector`

```

// This code creates the `DialogServiceConnector` with your subscription information.
// create a DialogServiceConfig by providing a Custom Commands application id and Cognitive Services
subscription key
// the RecoLanguage property is optional (default en-US); note that only en-US is supported in Preview
const string speechCommandsApplicationId = "YourApplicationId"; // Your application id
const string speechSubscriptionKey = "YourSpeechSubscriptionKey"; // Your subscription key
const string region = "YourServiceRegion"; // The subscription service region. Note: only 'westus2' is
currently supported

var speechCommandsConfig = CustomCommandsConfig.FromSubscription(speechCommandsApplicationId,
speechSubscriptionKey, region);
speechCommandsConfig SetProperty(PropertyId.SpeechServiceConnection_RecoLanguage, "en-us");
connector = new DialogServiceConnector(speechCommandsConfig);

```

4. Replace the strings `YourApplicationId`, `YourSpeechSubscriptionKey`, and `YourServiceRegion` with your own values for your app, speech subscription, and `region`

5. Append the following code snippet to the end of the method body of `InitializeDialogServiceConnector`

```

// This code sets up handlers for events relied on by `DialogServiceConnector` to communicate its
activities,
// speech recognition results, and other information.
//
// ActivityReceived is the main way your client will receive messages, audio, and events
connector.ActivityReceived += async (sender, activityReceivedEventArgs) =>
{
 NotifyUser(
 $"Activity received, hasAudio={activityReceivedEventArgs.HasAudio} activity=
{activityReceivedEventArgs.Activity}");

 if (activityReceivedEventArgs.HasAudio)
 {
 SynchronouslyPlayActivityAudio(activityReceivedEventArgs.Audio);
 }
};

// Canceled will be signaled when a turn is aborted or experiences an error condition
connector.Canceled += (sender, canceledEventArgs) =>
{
 NotifyUser($"Canceled, reason={canceledEventArgs.Reason}");
 if (canceledEventArgs.Reason == CancellationReason.Error)
 {
 NotifyUser(
 $"Error: code={canceledEventArgs.ErrorCode}, details={canceledEventArgs.ErrorDetails}");
 }
};

// Recognizing (not 'Recognized') will provide the intermediate recognized text
// while an audio stream is being processed
connector.Recognizing += (sender, recognitionEventArgs) =>
{
 NotifyUser($"Recognizing! in-progress text={recognitionEventArgs.Result.Text}");
};

// Recognized (not 'Recognizing') will provide the final recognized text
// once audio capture is completed
connector.Recognized += (sender, recognitionEventArgs) =>
{
 NotifyUser($"Final speech-to-text result: '{recognitionEventArgs.Result.Text}'");
};

// SessionStarted will notify when audio begins flowing to the service for a turn
connector.SessionStarted += (sender, sessionEventArgs) =>
{
 NotifyUser($"Now Listening! Session started, id={sessionEventArgs.SessionId}");
};

// SessionStopped will notify when a turn is complete and
// it's safe to begin listening again
connector.SessionStopped += (sender, sessionEventArgs) =>
{
 NotifyUser($"Listening complete. Session ended, id={sessionEventArgs.SessionId}");
};

```

6. Add the following code snippet to the body of the `ListenButton_ButtonClicked` method in the `MainPage` class

```

// This code sets up `DialogServiceConnector` to listen, since you already established the
configuration and
// registered the event handlers.
if (connector == null)
{
 InitializeDialogServiceConnector();
 // Optional step to speed up first interaction: if not called,
 // connection happens automatically on first use
 var connectTask = connector.ConnectAsync();
}

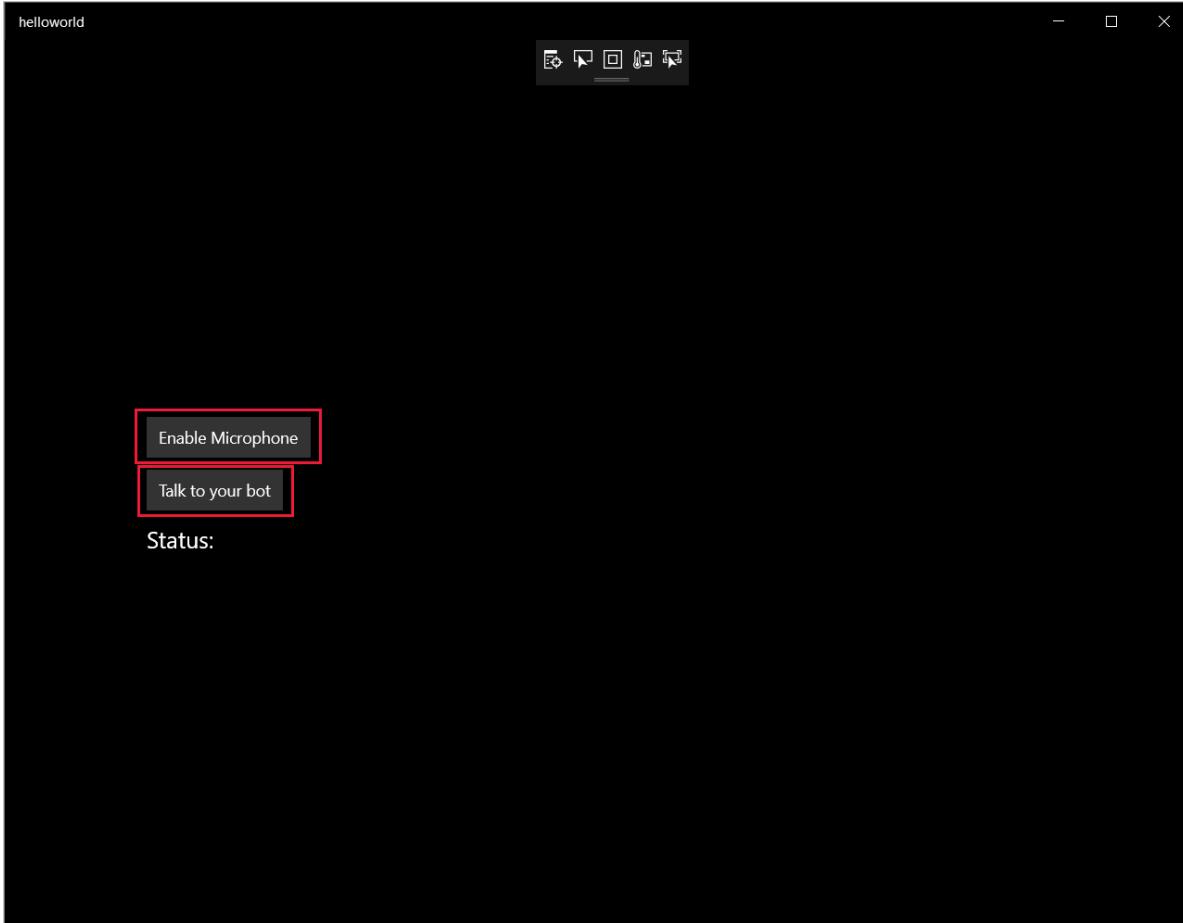
try
{
 // Start sending audio
 await connector.ListenOnceAsync();
}
catch (Exception ex)
{
 NotifyUser($"Exception: {ex.ToString()}", NotifyType.ErrorMessage);
}

```

- From the menu bar, choose **File > Save All** to save your changes

## Build and run the application

- From the menu bar, choose **Build > Build Solution** to build the application. The code should compile without errors.
- Choose **Debug > Start Debugging** (or press **F5**) to start the application. The **helloworld** window appears.



- Select **Enable Microphone**. If the access permission request pops up, select **Yes**.

Let helloworld access your microphone?

## Let helloworld access your microphone?

To change this later, go to the Settings app.

Yes

No

4. Select **Talk**, and speak an English phrase or sentence into your device's microphone. Your speech is transmitted to the Direct Line Speech channel and transcribed to text, which appears in the window.

## Next steps

[How to: Fulfill commands on the client with the Speech SDK \(preview\)](#) [How To: Add validations to Custom Command parameters \(Preview\)](#)

# How To: Fulfill Commands on the client with the Speech SDK (Preview)

1/8/2020 • 2 minutes to read • [Edit Online](#)

To complete tasks using a Custom Commands application you can send custom payloads to a connected client device.

In this article, you'll:

- Define and send a custom JSON payload from your Custom Commands application
- Receive and visualize the custom JSON payload contents from a C# UWP Speech SDK client application

## Prerequisites

- [Visual Studio 2019](#)
- An Azure subscription key for Speech service
  - [Get one for free](#) or create it on the [Azure portal](#)
- A previously created Custom Commands app
  - [Quickstart: Create a Custom Command with Parameters \(Preview\)](#)
- A Speech SDK enabled client application
  - [Quickstart: Connect to a Custom Command application with the Speech SDK \(Preview\)](#)

## Optional: Get started fast

This article describes, step by step, how to make a client application to talk to your Custom Commands application. If you prefer to dive right in, the complete, ready-to-compile source code used in this article is available in the [Speech SDK Samples](#).

## Fulfill with JSON payload

1. Open your previously created Custom Commands application from the [Speech Studio](#)
2. Check the **Completion Rules** section to make sure you have the previously created rule that responds back to the user
3. To send a payload directly to the client, create a new rule with a Send Activity action

# Completion Rules

Completion rules are executed once we're ready to fulfill, or complete, the command. They occur when all the required parameters are gathered. [Learn more](#)

## Rule Name

Name \*

## Conditions



RequiredParameters(OnOff, SubjectDevice)

## Actions



SendActivity (name, UpdateDeviceState)

**Save**

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	UpdateDeviceState	A name describing the purpose of the rule
Conditions	Required Parameter - <input type="text" value="onOff"/> and <input type="text" value="SubjectDevice"/>	Conditions that determine when the rule can run
Actions	<input type="text" value="SendActivity"/> (see below)	The action to take when the rule condition is true

Type \*

Send activity to client



#### Activity Content

```
1 {
2 "type": "event",
3 "name": "UpdateDeviceState",
4 "state": "{OnOff}",
5 "device": "{SubjectDevice}"
6 }
```

Cancel

Save

```
{
 "type": "event",
 "name": "UpdateDeviceState",
 "state": "{OnOff}",
 "device": "{SubjectDevice}"
}
```

## Create visuals for device on or off state

In [Quickstart: Connect to a Custom Command application with the Speech SDK \(Preview\)](#) you created a Speech SDK client application that handled commands such as `turn on the tv`, `turn off the fan`. Now add some visuals so you can see the result of those commands.

Add labeled boxes with text indicating **On** or **Off** using the following XML added to `MainPage.xaml.cs`

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" Margin="20">
 <Grid x:Name="Grid_TV" Margin="50, 0" Width="100" Height="100" Background="LightBlue">
 <StackPanel>
 <TextBlock Text="TV" Margin="0, 10" TextAlignment="Center"/>
 <TextBlock x:Name="State_TV" Text="Off" TextAlignment="Center"/>
 </StackPanel>
 </Grid>
 <Grid x:Name="Grid_Fan" Margin="50, 0" Width="100" Height="100" Background="LightBlue">
 <StackPanel>
 <TextBlock Text="Fan" Margin="0, 10" TextAlignment="Center"/>
 <TextBlock x:Name="State_Fan" Text="Off" TextAlignment="Center"/>
 </StackPanel>
 </Grid>
</StackPanel>
```

## Handle customizable payload

Now that you've created a JSON payload, you can add a reference to the [JSON.NET](#) library to handle deserialization.

A screenshot of the NuGet package manager interface. At the top, it says "NuGet: fulfillment" with a close button. Below that are tabs for "Browse", "Installed", and "Updates 1". A search bar contains the text "newtonsoft.json". To the right of the search bar are buttons for "X", a dropdown arrow, a refresh icon, and a checkbox labeled "Include prerelease". Below the search bar, there's a section for the "Newtonsoft.Json" package. It shows a rocket icon, the package name "Newtonsoft.Json" with a checkmark, the author "James Newton-King", "299M downloads", and a green circular badge with "v12.0.2". A description below says "Json.NET is a popular high-performance JSON framework for .NET".

In `InitializeDialogServiceConnector` add the following to your `ActivityReceived` event handler. The additional code will extract the payload from the activity and change the visual state of the tv or fan accordingly.

```
connector.ActivityReceived += async (sender, activityReceivedEventArgs) =>
{
 NotifyUser($"Activity received, hasAudio={activityReceivedEventArgs.HasAudio} activity=
{activityReceivedEventArgs.Activity}");

 dynamic activity = JsonConvert.DeserializeObject(activityReceivedEventArgs.Activity);

 if(activity?.name == "SetDeviceState")
 {
 var state = activity?.state;
 var device = activity?.device;
 switch(device)
 {
 case "tv":
 State_TV.Text = state;
 break;
 case "fan":
 State_Fan.Text = state;
 break;
 default:
 NotifyUser($"Received request to set unsupported device {device} to {state}");
 break;
 }
 }

 if (activityReceivedEventArgs.HasAudio)
 {
 SynchronouslyPlayActivityAudio(activityReceivedEventArgs.Audio);
 }
};
```

## Try it out

1. Start the application
2. Select Enable microphone
3. Select the Talk button
4. Say `turn on the tv`
5. The visual state of the tv should change to "On"

## Next steps

[How to: Add validations to Custom Command parameters \(preview\)](#)

# How To: Add validations to Custom Command parameters (Preview)

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

In this article, you'll learn how to add validations to parameters and prompt for correction.

## Prerequisites

You must have completed the steps in the following articles:

- [Quickstart: Create a Custom Command \(Preview\)](#)
- [Quickstart: Create a Custom Command with Parameters \(Preview\)](#)

## Create a SetTemperature Command

To demonstrate validations, let's create a new Command allowing the user to set the temperature.

1. Open your previously created Custom Commands application in [Speech Studio](#)
2. Create a new Command **SetTemperature**
3. Add a parameter for the target temperature
4. Add a validation for the temperature parameter

New Validation ×

Min Value \*

Max Value \*

Response Template \*

```
1 - Sorry, I can only set between 60 and 80 degrees
```

Create Cancel

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	Temperature	A descriptive name for your Command parameter
Required	true	Checkbox indicating whether a value for this parameter is required before completing the Command
Response template	"- What temperature would you like?"	A prompt to ask for the value of this parameter when it isn't known
Type	Number	The type of parameter, such as Number, String, or Date Time
Validation	Min Value: 60, Max Value: 80	For Number parameters, the allowed range of values for the parameter
Response template	"- Sorry, I can only set between 60 and 80 degrees"	Prompt to ask for an updated value if the validation fails

## 5. Add some sample sentences

```
set the temperature to {Temperature} degrees
change the temperature to {Temperature}
set the temperature
change the temperature
```

## 6. Add a Completion rule to confirm result

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	Confirmation Message	A name describing the purpose of the rule
Conditions	Required Parameter - Temperature	Conditions that determine when the rule can run
Actions	SpeechResponse - "- Ok, setting to {Temperature} degrees"	The action to take when the rule condition is true

### TIP

This example uses a speech response to confirm the result. For examples on completing the Command with a client action see: [How To: Fulfill Commands on the client with the Speech SDK \(Preview\)](#)

## Try it out

Select the Test panel and try a few interactions.

- Input: Set the temperature to 72 degrees
- Output: "Ok, setting to 72 degrees"
- Input: Set the temperature to 45 degrees

- Output: "Sorry, I can only set between 60 and 80 degrees"
- Input: make it 72 degrees instead
- Output: "Ok, setting to 72 degrees"

## Next steps

[How To: Add a confirmation to a Custom Command \(Preview\)](#)

# How To: Add a confirmation to a Custom Command (Preview)

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

In this article, you'll learn how to add a confirmation to a command.

## Prerequisites

You must have completed the steps in the following articles:

- [Quickstart: Create a Custom Command \(Preview\)](#)
- [Quickstart: Create a Custom Command with Parameters \(Preview\)](#)

## Create a SetAlarm command

To demonstrate validations, let's create a new Command allowing the user to set an alarm.

1. Open your previously created Custom Commands application in [Speech Studio](#)
2. Create a new Command **SetAlarm**
3. Add a parameter called DateTime

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	DateTime	A descriptive name for your Command parameter
Required	true	Checkbox indicating whether a value for this parameter is required before completing the Command
Response template	"- What time?"	A prompt to ask for the value of this parameter when it isn't known
Type	DateTime	The type of parameter, such as Number, String, or Date Time
Date Defaults	If date is missing use today	
Time Defaults	If time is missing use start of day	

4. Add some sample sentences

```
set an alarm for {DateTime}
set alarm {DateTime}
alarm for {DateTime}
```

5. Add a Completion rule to confirm result

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	Set alarm	A name describing the purpose of the rule
Actions	SpeechResponse - "- Ok, alarm set for {DateTime}"	The action to take when the rule condition is true

## Try it out

Select the Test panel and try a few interactions.

- Input: Set alarm for tomorrow at noon
- Output: "Ok, alarm set for 12/06/2019 12:00:00"
- Input: Set an alarm
- Output: "What time?"
- Input: 5pm
- Output: "Ok, alarm set for 12/05/2019 17:00:00"

## Add the advanced rules for confirmation

1. Add an advanced rule for confirmation.

This rule will ask the user to confirm the date and time of the alarm and is expecting a confirmation (yes/no) for the next turn.

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	Confirm date time	A name describing the purpose of the rule
Conditions	Required Parameter - DateTime	Conditions that determine when the rule can run
Actions	SpeechResponse - "- Are you sure you want to set an alarm for {DateTime}?"	The action to take when the rule condition is true
State after execution	Wait for input	State for the user after the turn
Expectations	Confirmation	Expectation for the next turn

2. Add an advanced rule to handle a successful confirmation (user said yes)

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	Accepted confirmation	A name describing the purpose of the rule
Conditions	SuccessfulConfirmation & Required Parameter - DateTime	Conditions that determine when the rule can run

SETTING	SUGGESTED VALUE	DESCRIPTION
State after execution	Ready for Completion	State of the user after the turn

3. Add an advanced rule to handle a confirmation denied (user said no)

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	Denied confirm	A name describing the purpose of the rule
Conditions	DeniedConfirmation & Required Parameter - DateTime	Conditions that determine when the rule can run
Actions	ClearParameter - DateTime & SpeechResponse - "- No problem, what time then?"	The action to take when the rule condition is true
State after execution	Wait for input	State of the user after the turn
Expectations	ElicitParameters - DateTime	Expectation for the next turn

## Try it out

Select the Test panel and try a few interactions.

- Input: Set alarm for tomorrow at noon
- Output: "Are you sure you want to set an alarm for 12/07/2019 12:00:00?"
- Input: No
- Output: "No problem, what time then?"
- Input: 5pm
- Output: "Are you sure you want to set an alarm for 12/06/2019 17:00:00?"
- Input: Yes
- Output: "Ok, alarm set for 12/06/2019 17:00:00"

## Next steps

[How To: Add a one-step correction to a Custom Command \(Preview\)](#)

# How To: Add a one-step correction to a Custom Command (Preview)

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

In this article, you'll learn how to add one-step confirmation to a command.

One-step correction is used to update a command that was just completed.

I.e. if you just set up an alarm, you can change your mind and update the time of the alarm.

- Input: Set alarm for tomorrow at noon
- Output: "Ok, alarm set for 12/06/2019 12:00:00"
- Input: No, tomorrow at 1pm
- Output: "Ok"

Keep in mind that this implies that you as a developer have a mechanism to update the alarm in your backend application.

## Prerequisites

You must have completed the steps in the following articles:

- [Quickstart: Create a Custom Command \(Preview\)](#)
- [Quickstart: Create a Custom Command with Parameters \(Preview\)](#)
- [How To: Add a confirmation to a Custom Command \(Preview\)](#)

## Add the advanced rules for one-step correction

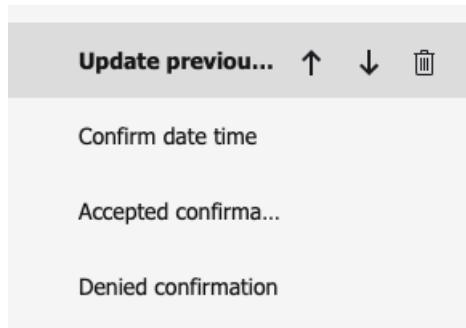
To demonstrate one-step correction, let's extend the **SetAlarm** command created in the [Confirmations How To](#).

1. Add an advanced rule to update the previous alarm.

This rule will ask the user to confirm the date and time of the alarm and is expecting a confirmation (yes/no) for the next turn.

SETTING	SUGGESTED VALUE	DESCRIPTION
Rule Name	Update previous alarm	A name describing the purpose of the rule
Conditions	UpdateLastCommand & Required Parameter - DateTime	Conditions that determine when the rule can run
Actions	SpeechResponse - "- Updating previous alarm to {DateTime}"	The action to take when the rule condition is true
State after execution	Complete command	State of the user after the turn

2. Move the rule you just created to the top of advanced rules (scroll over the rule in the panel and click the UP arrow).



#### NOTE

In a real application, in the Actions section of this rule you'll also send back an activity to the client or call an HTTP endpoint to update the alarm in your system.

## Try it out

Select the Test panel and try a few interactions.

- Input: Set alarm for tomorrow at noon
- Output: "Are you sure you want to set an alarm for 12/07/2019 12:00:00?"
- Input: Yes
- Output: "Ok, alarm set for 12/07/2019 12:00:00"
- Input: No, tomorrow at 1pm
- Output: "Updating previous alarm to 12/07/2019 13:00:00"

# Tutorial: Voice-enable your bot using the Speech SDK

12/26/2019 • 20 minutes to read • [Edit Online](#)

You can now use the power of the Speech service to easily voice-enable a chat bot.

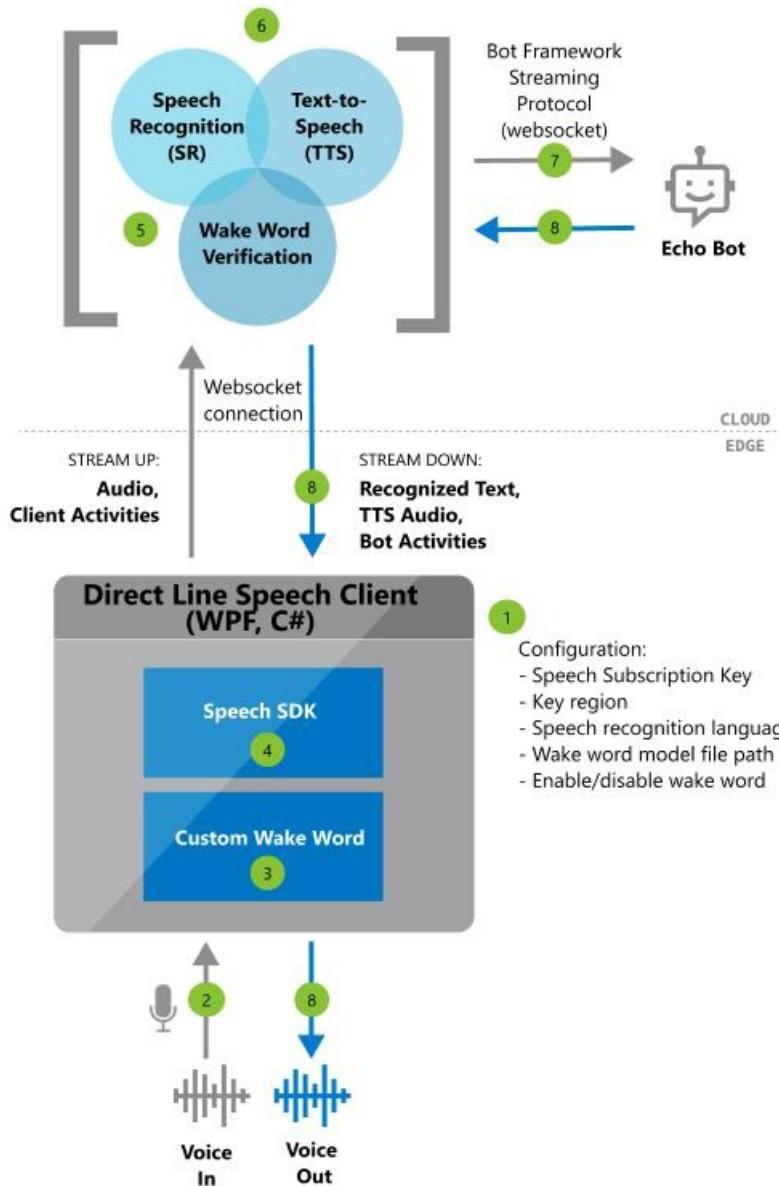
In this tutorial, you'll create an Echo Bot using Microsoft Bot-Framework, deploy it to Azure, and register it with the Bot-Framework Direct Line Speech channel. Then you'll configure a sample client app for Windows that lets you speak to your bot and hear it respond back to you.

This tutorial is designed for developers who are just starting their journey with Azure, Bot-Framework bots, Direct Line Speech, or the Speech SDK, and want to quickly build a working system with limited coding. No experience or familiarity with these services is needed.

At the end of this exercise, you'll have set up a system that will operate as follows:

1. The sample client application is configured to connect to Direct Line Speech channel and the Echo Bot
2. Audio is recorded from the default microphone on button press (or continuously recorded if custom keyword is activated)
3. Optionally, custom keyword detection happens, gating audio streaming to the cloud
4. Using Speech SDK, the app connects to Direct Line Speech channel and streams audio
5. Optionally, higher accuracy keyword verification happens on the service
6. The audio is passed to the speech recognition service and transcribed to text
7. The recognized text is passed to the Echo-Bot as a Bot Framework Activity
8. The response text is turned into audio by the Text-to-Speech (TTS) service, and streamed back to the client application for playback

## Direct Line Speech channel Co-located services



### NOTE

The steps in this tutorial do not require a paid service. As a new Azure user, you'll be able to use credits from your free Azure trial subscription and the free tier of the Speech service to complete this tutorial.

Here's what this tutorial covers:

- Create new Azure resources
- Build, test, and deploy the Echo Bot sample to an Azure App Service
- Register your bot with Direct Line Speech channel
- Build and run the Direct Line Speech Client to interact with your Echo Bot
- Add custom keyword activation
- Learn to change the language of the recognized and spoken speech

## Prerequisites

Here's what you'll need to complete this tutorial:

- A Windows 10 PC with a working microphone and speakers (or headphones)

- [Visual Studio 2017](#) or higher
- [.NET Core SDK](#) version 2.1 or later
- An Azure account. [Sign up for free](#).
- A [GitHub](#) account
- [Git for Windows](#)

## Create a resource group

The client app that you'll create in this tutorial uses a handful of Azure services. To reduce the round-trip time for responses from your bot, you'll want to make sure that these services are located in the same Azure region. In this section, you'll create a resource group in the **West US** region. This resource group will be used when creating individual resources for the Bot-Framework, the Direct Line Speech channel, and the Speech service.

1. Sign in to the [Azure portal](#).
2. From the left navigation, select **Resource groups**. Then click **Add** to add a new resource group.
3. You'll be prompted to provide some information:
  - Set **Subscription** to **Free Trial** (you can also use an existing subscription).
  - Enter a name for your **Resource group**. We recommend **SpeechEchoBotTutorial-ResourceGroup**.
  - From the **Region** drop-down, select **West US**.
4. Click **Review and create**. You should see a banner that read **Validation passed**.
5. Click **Create**. It may take a few minutes to create the resource group.
6. As with the resources you'll create later in this tutorial, it's a good idea to pin this resource group to your dashboard for easy access. If you'd like to pin this resource group, click the pin icon in the upper right of the dashboard.

### Choosing an Azure region

If you'd like to use a different region for this tutorial these factors may limit your choices:

- Ensure that you use a [supported Azure region](#).
- The Direct Line Speech channel uses the text-to-speech service, which has standard and neural voices. Neural voices are [limited to specific Azure regions](#).
- Free trial keys may be restricted to a specific region.

For more information about regions, see [Azure locations](#).

## Create resources

Now that you have a resource group in the **West US** region, the next step is to create individual resources for each service that you'll use in this tutorial.

### Create a Speech service resource

Follow these instructions to create a Speech resource:

1. Go to the [Azure portal](#) and select **Create a resource** from the left navigation.
2. In the search bar, type **Speech**.
3. Select **Speech**, then click **Create**.
4. You'll be prompted to provide some information:
  - Give your resource a **Name**. We recommend **SpeechEchoBotTutorial-Speech**
  - For **Subscription**, make sure **Free Trial** is selected.
  - For **Location**, select **West US**.
  - For **Pricing tier**, select **F0**. This is the free tier.
  - For **Resource group**, select **SpeechEchoBotTutorial-ResourceGroup**.

- After you've entered all required information, click **Create**. It may take a few minutes to create your resource.
- Later in this tutorial you'll need subscription keys for this service. You can access these keys at any time from your resource's **Overview** (Manage keys) or **Keys**.

At this point, check that your resource group (**SpeechEchoBotTutorial-ResourceGroup**) has a Speech resource:

NAME	TYPE	LOCATION
SpeechEchoBotTutorial-Speech	Cognitive Services	West US

### Create an Azure App Service plan

The next step is to create an App Service Plan. An App Service plan defines a set of compute resources for a web app to run.

- Go to the [Azure portal](#) and select **Create a resource** from the left navigation.
- In the search bar, type **App Service Plan**. Next, locate and select the **App Service Plan** card from the search results.
- Click **Create**.
- You'll be prompted to provide some information:
  - Set **Subscription** to **Free Trial** (you can also use an existing subscription).
  - For **Resource group**, select **SpeechEchoBotTutorial-ResourceGroup**.
  - Give your resource a **Name**. We recommend **SpeechEchoBotTutorial-AppServicePlan**
  - For **Operating System**, select **Windows**.
  - For **Region**, select **West US**.
  - For **Pricing Tier**, make sure that **Standard S1** is selected. This should be the default value. If it isn't, make sure you set the **Operating System** to **Windows** as described above.
- Click **Review and create**. You should see a banner that read **Validation passed**.
- Click **Create**. It may take a few minutes to create the resource group.

At this point, check that your resource group (**SpeechEchoBotTutorial-ResourceGroup**) has two resources:

NAME	TYPE	LOCATION
SpeechEchoBotTutorial-AppServicePlan	App Service Plan	West US
SpeechEchoBotTutorial-Speech	Cognitive Services	West US

## Build an Echo Bot

Now that you've created some resources, let's build a bot. We're going to start with the Echo Bot sample, which as the name implies, echoes the text that you've entered as its response. Don't worry, the sample code is ready for you to use without any changes. It's configured to work with the Direct Line Speech channel, which we'll connect after we've deployed the bot to Azure.

#### NOTE

The instructions that follow, as well as additional information about the Echo Bot are available in the [sample's README on GitHub](#).

### Run the bot sample on your machine

1. Clone the samples repository:

```
git clone https://github.com/Microsoft/botbuilder-samples.git
```

2. Launch Visual Studio.

3. From the toolbar, select **File > Open > Project/Solution**, and open the Echo Bot project solution:

```
samples\csharp_dotnetcore\02.echo-bot\EchoBot.sln
```

4. After the project is loaded, press **F5** to build and run the project.

#### Test the bot sample with the Bot Framework Emulator

The [Bot Framework Emulator](#) is a desktop app that allows bot developers to test and debug their bots locally or remotely through a tunnel. The Emulator supports typed text as the input (not voice). The bot will response with text. Follow these steps to use the Bot Framework Emulator to test your Echo Bot running locally, with text input and text output. After we deploy the bot Azure we will test it with voice input and voice output.

1. Install the [Bot Framework Emulator](#) version 4.3.0 or greater

2. Launch the Bot Framework Emulator and open your bot:

- **File -> Open Bot.**

3. Enter the URL for your bot. For example:

```
http://localhost:3978/api/messages
```

and press "Connect".

4. The bot should immediately greet you with "Hello and welcome!" message. Type in any text message and confirm you get a response from the bot.

## Deploy your bot to an Azure App Service

The next step is to deploy the Echo Bot to Azure. There are a few ways to deploy a bot, but in this tutorial we'll focus on publishing directly from Visual Studio.

#### NOTE

Alternatively, you can deploy a bot using the [Azure CLI](#) and [deployment templates](#).

1. From Visual Studio, open the Echo Bot that's been configured for use with Direct Line Speech channel:

```
samples\csharp_dotnetcore\02.echo-bot\EchoBot.sln
```

2. In the **Solution Explorer**, right-click the **EchoBot** project and select **Publish...**

3. A new window titled **Pick a publish target** will open.

4. Select **App Service** from the left navigation, select **Create New**, then click **Publish**.

5. When the **Create App Service** window appears:

- Click **Add an account**, and sign in with your Azure account credentials. If you're already signed in,

select the account you want from the drop-down list.

- For the **App Name**, you'll need to enter a globally unique name for your Bot. This name is used to create a unique bot URL. A default value will be populated including the date and time (For example: "EchoBot20190805125647"). You can use the default name for this tutorial.
- For **Subscription**, set it to **Free Trial**
- For **Resource Group**, select **SpeechEchoBotTutorial-ResourceGroup**
- For **Hosting Plan**, select **SpeechEchoBotTutorial-AppServicePlan**

6. Click **Create**

7. You should see a success message in Visual Studio that looks like this:

```
Publish Succeeded.
Web App was published successfully https://EchoBot20190805125647.azurewebsites.net/
```

8. Your default browser should open and display a page that reads: "Your bot is ready!".

9. At this point, check your Resource Group **SpeechEchoBotTutorial-ResourceGroup** in the Azure portal, and confirm there are three resources:

NAME	TYPE	LOCATION
EchoBot20190805125647	App Service	West US
SpeechEchoBotTutorial-AppServicePlan	App Service plan	West US
SpeechEchoBotTutorial-Speech	Cognitive Services	West US

## Enable web sockets

You'll need to make a small configuration change so that your bot can communicate with the Direct Line Speech channel using web sockets. Follow these steps to enable web sockets:

1. Navigate to the [Azure portal](#), and locate your App Service. The resource should be named similar to **EchoBot20190805125647** (your unique app name).
2. In the left navigation, under **Settings**, click **Configuration**.
3. Select the **General settings** tab.
4. Locate the toggle for **Web sockets** and set it to **On**.
5. Click **Save**.

**TIP**

You can use the controls at the top of your Azure App Service page to stop or restart the service. This may come in handy when troubleshooting.

## Create a channel registration

Now that you've created an Azure App Service to host your bot, the next step is to create a **Bot Channels Registration**. Creating a channel registration is a prerequisite for registering your bot with Bot-Framework channels, including Direct Line Speech channel.

#### NOTE

If you'd like to learn more about how bots leverage channels, see [Connect a bot to channels](#).

1. The first step is to create a new resource for the registration. In the [Azure portal](#), click **Create a resource**.
2. In the search bar type **bot**, after the results appear, select **Bot Channels Registration**.
3. Click **Create**.
4. You'll be prompted to provide some information:
  - For **Bot handle**, enter **SpeechEchoBotTutorial-BotRegistration**.
  - For **Subscription**, select **Free Trial**.
  - For **Resource group**, select **SpeechEchoBotTutorial-ResourceGroup**.
  - For **Location**, select **West US**.
    - For **Pricing tier**, select **F0**.
    - For **Messaging endpoint**, enter the URL for your web app with the `/api/messages` path appended at the end. For example: if your globally unique App Name was **EchoBot20190805125647**, your messaging endpoint would be:  
`https://EchoBot20190805125647.azurewebsites.net/api/messages/`.
    - For **Application insights**, you can set this to **Off**. For more information, see [Bot analytics](#).
    - Ignore **Auto create App ID and password**.
5. At the bottom of the **Bot Channels Registration** blade, click **Create**.

At this point, check your Resource Group **SpeechEchoBotTutorial-ResourceGroup** in the Azure portal. It should now show four resources:

NAME	TYPE	LOCATION
EchoBot20190805125647	App Service	West US
SpeechEchoBotTutorial-AppServicePlan	App Service plan	West US
SpeechEchoBotTutorial-BotRegistration	Bot Channels Registration	Global
SpeechEchoBotTutorial-Speech	Cognitive Services	West US

#### IMPORTANT

The Bot Channels Registration resource will show the Global region even though you selected West US. This is expected.

## Register the Direct Line Speech channel

Now it's time to register your bot with the Direct Line Speech channel. This channel is what's used to create a connection between your echo bot and a client app compiled with the Speech SDK.

1. Locate and open your **SpeechEchoBotTutorial-BotRegistration** resource in the [Azure portal](#).
2. From the left navigation, select **Channels**.
  - Look for **More channels**, locate and click **Direct Line Speech**.
  - Review the text on the page titled **Configure Direct line Speech**, then expand the drop-down menu labeled "Cognitive service account."
  - Select the speech resource you created earlier (e.g., **SpeechEchoBotTutorial-Speech**) from the menu

to associate your bot to your speech subscription key.

- Click **Save**.
3. From the left navigation, click **Settings**.
- Check the box labeled **Enable Streaming Endpoint**. This is needed to enable a communication protocol built on web sockets between your bot and the Direct Line Speech channel.
  - Click **Save**.

**TIP**

If you'd like to learn more, see [Connect a bot to Direct Line Speech](#). This page includes additional information and known issues.

## Build the Direct Line Speech Client

In this step, you're going to build the Direct Line Speech Client. The client is a Windows Presentation Foundation (WPF) app in C# that uses the [Speech SDK](#) to manage communication with your bot using the Direct Line Speech channel. Use it to interact with and test your bot before writing a custom client app.

The Direct Line Speech Client has a simple UI that allows you to configure the connection to your bot, view the text conversation, view Bot-Framework activities in JSON format, and display adaptive cards. It also supports the use of custom keywords. You'll use this client to speak with your bot and receive a voice response.

Before we move on, make sure that your microphone and speakers are enabled and working.

1. Navigate to the GitHub repository for the [Direct Line Speech Client](#).
2. Follow the instructions provided to clone the repository, build the project, configure the client, and launch the client.
3. Click **Reconnect** and make sure you see the message **Press the mic button, or type to start talking to your bot**.
4. Let's test it out. Click the microphone button, and speak a few words in English. The recognized text will appear as you speak. When you're done speaking, the bot will reply in its own voice, saying "echo" followed by the recognized words.
5. You can also use text to communicate with the bot. Just type in the text at the bottom bar.

### Troubleshooting errors in Direct Line Speech Client

If you get an error message in your main app window, use this table to identify and troubleshoot the error:

ERROR	WHAT SHOULD YOU DO?
Error AuthenticationFailure: WebSocket Upgrade failed with an authentication error (401). Check for correct subscription key (or authorization token) and region name	In the Settings page of the app, make sure you entered the Speech Subscription key and its region correctly. Make sure your speech key and key region were entered correctly.
Error ConnectionFailure: Connection was closed by the remote host. Error code: 1011. Error details: We could not connect to the bot before sending a message	Make sure you <a href="#">checked the "Enable Streaming Endpoint" box</a> and/or <a href="#">toggled Web sockets to On</a> . Make sure your Azure App Service is running. If it is, try restarting your App Service.
Error ConnectionFailure: Connection was closed by the remote host. Error code: 1011. Error details: Response status code does not indicate success: 500 (InternalServerError)	Your bot specified a neural voice in its output Activity <a href="#">Speak</a> field, but the Azure region associated with your Speech subscription key does not support neural voices. See <a href="#">Standard and neural voices</a> .

ERROR	WHAT SHOULD YOU DO?
Error ConnectionFailure: Connection was closed by the remote host. Error code: 1000. Error details: Exceeded maximum web socket connection idle duration(> 300000 ms)	This is an expected error when a connection to the channel is left open and inactive for more than five minutes.

If your issue isn't addressed in the table, see [Voice assistants: Frequently asked questions](#).

## View bot activities

Every bot sends and receives **Activity** messages. In the **Activity Log** window of Direct Line Speech Client, you'll see timestamped logs with each activity that the client has received from the bot. You can also see the activities that the client sent to the bot using the `DialogServiceConnector.SendActivityAsync` method. When you select a log item, it will show the details of the associated activity as JSON.

Here's a sample json of an Activity the client received:

```
{
 "attachments": [],
 "channelData": {
 "conversationalAiData": {
 "requestInfo": {
 "interactionId": "8d5cb416-73c3-476b-95fd-9358cbfaebfa",
 "version": "0.2"
 }
 },
 "channelId": "directlinespeech",
 "conversation": {
 "id": "129ebffe-772b-47f0-9812-7c5bfd4aca79",
 "isGroup": false
 },
 "entities": [],
 "from": {
 "id": "SpeechEchoBotTutorial-BotRegistration"
 },
 "id": "89841b4d-46ce-42de-9960-4fe4070c70cc",
 "inputHint": "acceptingInput",
 "recipient": {
 "id": "129ebffe-772b-47f0-9812-7c5bfd4aca79|0000"
 },
 "replyToId": "67c823b4-4c7a-4828-9d6e-0b84fd052869",
 "serviceUrl": "urn:botframework:websocket:directlinespeech",
 "speak": "<speak version='1.0' xmlns='https://www.w3.org/2001/10/synthesis' xml:lang='en-US'><voice name='Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)'>Echo: Hello and welcome.</voice></speak>",
 "text": "Echo: Hello and welcome.",
 "timestamp": "2019-07-19T20:03:51.1939097Z",
 "type": "message"
 }
}
```

To learn more about what's returned in the JSON output, see [fields in the Activity](#). For the purpose of this tutorial, you can focus on the [Text](#) and [Speak](#) fields.

## View client source code for calls to the Speech SDK

The Direct Line Speech Client uses the NuGet package [Microsoft.CognitiveServices.Speech](#), which contains the Speech SDK. A good place to start reviewing the sample code is the method `InitSpeechConnector()` in file

`DLSpeechClient\MainWindow.xaml.cs`, which creates these two Speech SDK objects:

- `DialogServiceConfig` - For configuration settings (e.g., speech subscription key, key region)
- `DialogServiceConnector` - To manage the channel connection and client subscription events for handling

recognized speech and bot responses.

## Add custom keyword activation

The Speech SDK supports custom keyword activation. Similar to "Hey Cortana" for Microsoft's Assistant, you can write an app that will continuously listen for a keyword of your choice. Keep in mind that a keyword can be single word or a multi-word phrase.

### NOTE

The term *keyword* is often used interchangeably with the term *wake word*, and you may see both used in Microsoft documentation.

Keyword detection is done on the client app. If using a keyword, audio is only streamed to the Direct Line Speech channel if the keyword is detected. The Direct Line Speech channel includes a component called *keyword verification (KVV)*, which does more complex processing in the cloud to verify that the keyword you've chosen is at the start of the audio stream. If key word verification succeeds, then the channel will communicate with the bot.

Follow these steps to create a keyword model, configure the Direct Line Speech Client to use this model, and finally, test it with your bot.

1. Follow these instructions to [create a custom keyword by using the Speech service](#).
2. Unzip the model file that you downloaded in the previous step. It should be named for your keyword. You're looking for a file named `kws.table`.
3. In the Direct Line Speech client, locate the **Settings** menu (look for the gear icon in the top right). Locate **Model file path** and enter the full path name for the `kws.table` file from step 2.
4. Make sure to check the box labeled **Enabled**. You should see this message next to the check box: "Will listen for the keyword upon next connection". If you've provided the wrong file or an invalid path, you should see an error message.
5. Enter your speech **subscription key**, **subscription key region**, and then click **OK** to close the **Settings** menu.
6. Click **Reconnect**. You should see a message that reads: "New conversation started - type, press the microphone button, or say the keyword". The app is now continuously listening.
7. Speak any phrase that starts with your keyword. For example: "**{your keyword}**, what time is it?". You don't need to pause after uttering the keyword. When finished, two things happen:
  - You'll see a transcription of what you spoke
  - Shortly after, you'll hear the bot's response
8. Continue to experiment with the three input types that your bot supports:
  - Typed-text in the bottom bar
  - Pressing the microphone icon and speaking
  - Saying a phrase that starts with your keyword

### View the source code that enables keyword

In the Direct Line Speech Client source code, take a look at these files to review the code that's used to enable keyword detection:

1. `DLSpeechClient\Models.cs` includes a call to the Speech SDK method `KeywordRecognitionModel.fromFile()`, which is used to instantiate the model from a local file on disk.
2. `DLSpeechClient\MainWindow.xaml.cs` includes a call to Speech SDK method `DialogServiceConnector.StartKeywordRecognitionAsync()`, which activates continuous keyword detection.

# (Optional) Change the language and bot voice

The bot that you've created will listen for and respond in English, with a default American English text-to-speech voice. However, you're not limited to using English, or a default voice. In this section, you'll learn how to change the language that your bot will listen for and respond in. You will also learn how to select a different voice for that language.

## Change the language

You can chose from any one of the languages mentioned in the [speech-to-text](#) table. In the example below, we will change the language to German.

1. Open the Direct Line Speech Client app, click on the settings button (upper-right gear icon), and enter `de-de` in the Language field (this is the Locale value mentioned in the [speech-to-text](#) table). This sets the spoken language to be recognized, overriding the default `en-us`. This also instructs Direct Line Speech channel to use a default German voice for the Bot reply.
2. Close the settings page, and click on the Reconnect button to establish a new connection to your echo bot.
3. Click on the microphone button, and say a phrase in German. You will see the recognized text and the echo bot replying with the default German voice.

## Change the default bot voice

Selecting the text-to-speech voice and controlling pronunciation can be done if the Bot specifies the reply in the form of a [Speech Synthesis Markup Language \(SSML\)](#) instead of simple text. The echo bot does not use SSML, but we can easily modify the code to do that. In the example below we add SSML to the echo bot reply, such that the German voice Stefan Apollo (a male voice) will be used instead of the default female voice. See list of [Standard Voices](#) and [Neural Voices](#) supported for your language.

1. Let's start by opening `samples\csharp_dotnetcore\02.echo-bot\echo-bot.cs`.
2. Locate these two lines:

```
var replyText = $"Echo: {turnContext.Activity.Text}";
await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replyText), cancellationToken);
```

3. Replace them with:

```
var replyText = $"Echo: {turnContext.Activity.Text}";
var replySpeak = @"<speak version='1.0' xmlns='https://www.w3.org/2001/10/synthesis' xml:lang='de-DE'>
 <voice name='Microsoft Server Speech Text to Speech Voice (de-DE, Stefan, Apollo)'>" +
 $"{replyText}" + "</voice></speak>";
await turnContext.SendActivityAsync(MessageFactory.Text(replyText, replySpeak), cancellationToken);
```

4. Build your solution in Visual Studio and fix any build errors.

The second argument in the method 'MessageFactory.Text' sets the [Activity speak field](#) in the bot reply. With the above change, it has been replaced from simple text to SSML in order to specify a non-default German voice.

## R redeploy your bot

Now that you've made the necessary change to the bot, the next step is to republish it to your Azure App Service and try it out:

1. In the Solution Explorer window, right-click on the **EchoBot** project and select **Publish**.
2. Your previous deployment configuration has already been loaded as the default. Simply click **Publish** next to **EchoBot20190805125647 - Web Deploy**.
3. The **Publish Succeeded** message will appear in the Visual Studio output window, and a web page will launch with the message "Your bot is ready!".

4. Open the Direct Line Speech Client app, click on the settings button (upper-right gear icon), and make sure you still have `de-de` in the Language field.
5. Follow the instructions in [Build the Direct Line Speech Client](#) to reconnect with your newly deployed bot, speak in the new language and hear your bot reply in that language with the new voice.

## Clean up resources

If you're not going to continue using the echo-bot deployed in this tutorial, you can remove it and all its associated Azure resources by simply deleting the Azure Resource group **SpeechEchoBotTutorial-ResourceGroup**.

1. From the [Azure portal](#), click on **Resource Groups** from the left navigation.
2. Find the resource group named: **SpeechEchoBotTutorial-ResourceGroup**. Click the three dots (...).
3. Select **Delete resource group**.

## Next steps

[Build your own client app with the Speech SDK](#)

## See also

- Deploying to an [Azure region near you](#) to see bot response time improvement
- Deploying to an [Azure region that supports high quality Neural TTS voices](#)
- Pricing associated with Direct Line Speech channel:
  - [Bot Service pricing](#)
  - [Speech service](#)
- Building and deploying your own voice-enabled bot:
  - Build a [Bot-Framework bot](#). Register it with [Direct Line Speech channel](#) and [customize your bot for voice](#)
  - Explore existing [Bot-Framework solutions](#): Build a [virtual assistant](#) and [extend it to Direct Line Speech](#)

# Voice assistants frequently asked questions

11/14/2019 • 3 minutes to read • [Edit Online](#)

If you can't find answers to your questions in this document, check out [other support options](#).

## General

### **Q: What is a voice assistant?**

**A:** Like Cortana, a voice assistant is a solution that listens to a user's spoken utterances, analyzes the contents of those utterances for meaning, performs one or more actions in response to the utterance's intent, and then provides a response to the user that often includes a spoken component. It's a "voice-in, voice-out" experience for interacting with a system. voice assistant authors create an on-device application using the `DialogServiceConnector` in the Speech SDK to communicate with an assistant created using [Custom Commands \(Preview\)](#) or the [Direct Line Speech](#) channel of the Bot Framework. These assistants can use custom keywords, custom speech, and custom voice to provide an experience tailored to your brand or product.

### **Q: Should I use Custom Commands (Preview) or Direct Line Speech? What's the difference?**

**A:** [Custom Commands \(Preview\)](#) is a lower-complexity set of tools to easily create and host an assistant that's well-suited to task completion scenarios. [Direct Line Speech](#) provides richer, more sophisticated capabilities that can enable robust conversational scenarios. See the [comparison of assistant solutions](#) for more information.

### **Q: How do I get started?**

**A:** The best way to begin with creating a Custom Commands (Preview) application or basic Bot Framework bot.

- [Create a Custom Commands \(Preview\) application](#)
- [Create a basic Bot Framework bot](#)
- [Connect a bot to the Direct Line Speech channel](#)

## Debugging

### **Q: Where's my channel secret?**

**A:** If you've used the preview version of Direct Line Speech or you're reading related documentation, you may expect to find a secret key on the Direct Line Speech channel registration page. The v1.7 `DialogServiceConfig` factory method `FromBotSecret` in the Speech SDK also expects this value.

The latest version of Direct Line Speech simplifies the process of contacting your bot from a device. On the channel registration page, the drop-down at the top associates your Direct Line Speech channel registration with a speech resource. Once associated, the v1.8 Speech SDK includes a `BotFrameworkConfig::FromSubscription` factory method that will configure a `DialogServiceConnector` to contact the bot you've associated with your subscription.

If you're still migrating your client application from v1.7 to v1.8, `DialogServiceConfig::FromBotSecret` may continue to work with a non-empty, non-null value for its channel secret parameter, e.g. the previous secret you used. It will simply be ignored when using a speech subscription associated with a newer channel registration. Please note that the value *must* be non-null and non-empty, as these are checked for on the device before the service-side association is relevant.

For a more detailed guide, please see the [tutorial section](#) that walks through channel registration.

### **Q: I get a 401 error when connecting and nothing works. I know my speech subscription key is valid.**

## What's going on?

**A:** When managing your subscription on the Azure portal, please ensure you're using the **Speech** resource (Microsoft.CognitiveServicesSpeechServices, "Speech") and *not* the **Cognitive Services** resource (Microsoft.CognitiveServicesAllInOne, "All Cognitive Services"). Also, please check [Speech service region support for voice assistants](#).

MySpeechSubscription  
Cognitive Services

Search (Ctrl+)

Unavailable setting Delete

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Resource group (change) : MyResourceGroup  
Status : Active  
Location : West US 2 (highlighted)  
Subscription (change) : My Subscription  
Subscription ID : My-Subscription-Id  
Tags (change) : Click here to add tags

API type : Speech (highlighted)  
Pricing tier : Standard  
Endpoint : https://westus2.api.cognitive.microsoft.com/sts/v1.0/issuetoken  
Manage keys : Show access keys ...

**Q:** I get recognition text back from my `DialogServiceConnector`, but I see a '1011' error and nothing from my bot. Why?

**A:** This error indicates a communication problem between your assistant and the voice assistant service.

- For Custom Commands (Preview), ensure that your Custom Commands (Preview) Application is published
- For Direct Line Speech, ensure that you've [connected your bot to the Direct Line Speech channel](#), [added Streaming protocol support](#) to your bot (with the related Web Socket support), and then check that your bot is responding to incoming requests from the channel.

**Q:** This code still doesn't work and/or I'm getting a different error when using a `DialogServiceConnector`. What should I do?

**A:** File-based logging provides substantially more detail and can help accelerate support requests. To enable this functionality, see [how to use file logging](#).

## Next steps

- [Troubleshooting](#)
- [Release notes](#)

# What is Custom Speech?

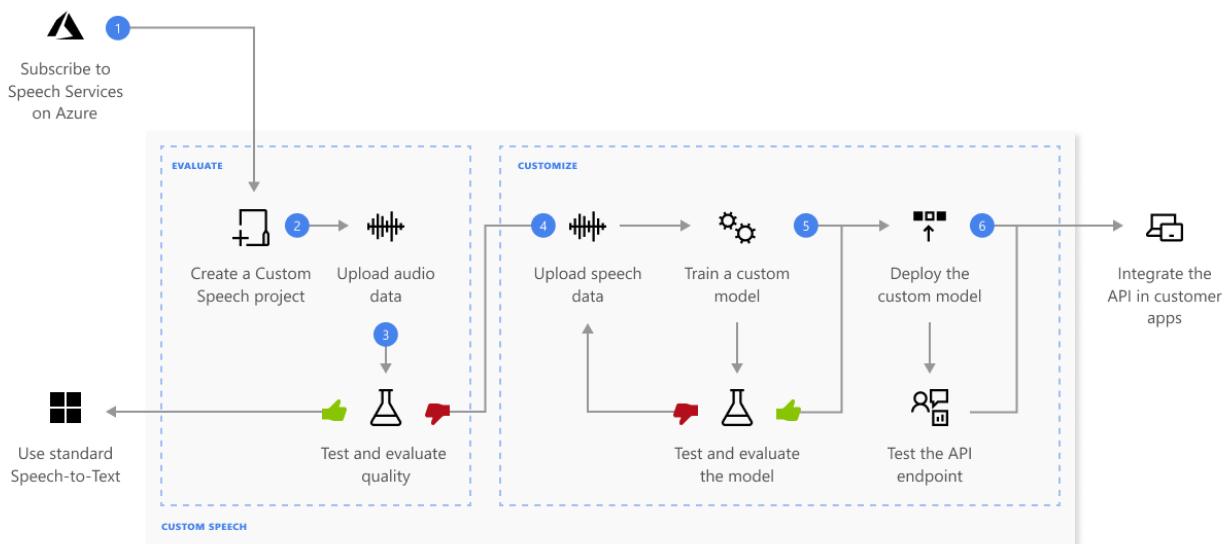
12/4/2019 • 3 minutes to read • [Edit Online](#)

**Custom Speech** is a set of online tools that allow you to evaluate and improve Microsoft's speech-to-text accuracy for your applications, tools, and products. All it takes to get started are a handful of test audio files. Follow the links below to start creating a custom speech-to-text experience.

## What's in Custom Speech?

Before you can do anything with Custom Speech, you'll need an Azure account and a Speech service subscription. Once you've got an account, you can prep your data, train and test your models, inspect recognition quality, evaluate accuracy, and ultimately deploy and use the custom speech-to-text model.

This diagram highlights the pieces that make up the [Custom Speech portal](#). Use the links below to learn more about each step.



1. **Subscribe and create a project** - Create an Azure account and subscribe to the Speech service. This unified subscription gives you access to speech-to-text, text-to-speech, speech translation, and the [Custom Speech portal](#). Then, using your Speech service subscription, create your first Custom Speech project.
2. **Upload test data** - Upload test data (audio files) to evaluate Microsoft's speech-to-text offering for your applications, tools, and products.
3. **Inspect recognition quality** - Use the [Custom Speech portal](#) to play back uploaded audio and inspect the speech recognition quality of your test data. For quantitative measurements, see [Inspect data](#).
4. **Evaluate accuracy** - Evaluate the accuracy of the speech-to-text model. The [Custom Speech portal](#) will provide a *Word Error Rate*, which can be used to determine if additional training is required. If you're satisfied with the accuracy, you can use the Speech service APIs directly. If you'd like to improve accuracy by a relative average of 5% - 20%, use the **Training** tab in the portal to upload additional training data, such as human-labeled transcripts and related text.
5. **Train the model** - Improve the accuracy of your speech-to-text model by providing written transcripts (10-1,000 hours) and related text (<200 MB) along with your audio test data. This data helps to train the speech-to-text model. After training, retest, and if you're satisfied with the result, you can deploy your model.

6. [Deploy the model](#) - Create a custom endpoint for your speech-to-text model and use it in your applications, tools, or products.

## Set up your Azure account

A Speech service subscription is required before you can use the [Custom Speech portal](#) to create a custom model. Follow these instructions to create a standard Speech service subscription: [Create a Speech Subscription](#).

### NOTE

Please be sure to create standard (S0) subscriptions, free trial (F0) subscriptions are not supported.

Once you've created an Azure account and a Speech service subscription, you'll need to sign in to [Custom Speech portal](#) and connect your subscription.

1. Get your Speech service subscription key from the Azure portal.
2. Sign-in to the [Custom Speech portal](#).
3. Select the subscription you need to work on and create a speech project.
4. If you'd like to modify your subscription, use the **cog** icon located in the top navigation.

## How to create a project

Content like data, models, tests, and endpoints are organized into **Projects** in the [Custom Speech portal](#). Each project is specific to a domain and country/language. For example, you may create a project for call centers that use English in the United States.

To create your first project, select the **Speech-to-text/Custom speech**, then click **New Project**. Follow the instructions provided by the wizard to create your project. After you've created a project, you should see four tabs: **Data, Testing, Training, and Deployment**. Use the links provided in [Next steps](#) to learn how to use each tab.

## Next steps

- [Prepare and test your data](#)
- [Inspect your data](#)
- [Evaluate your data](#)
- [Train your model](#)
- [Deploy your model](#)

# Prepare data for Custom Speech

1/3/2020 • 7 minutes to read • [Edit Online](#)

When testing the accuracy of Microsoft speech recognition or training your custom models, you'll need audio and text data. On this page, we cover the types of data, how to use, and manage them.

## Data types

This table lists accepted data types, when each data type should be used, and the recommended quantity. Not every data type is required to create a model. Data requirements will vary depending on whether you're creating a test or training a model.

DATA TYPE	USED FOR TESTING	RECOMMENDED QUANTITY	USED FOR TRAINING	RECOMMENDED QUANTITY
Audio	Yes Used for visual inspection	5+ audio files	No	N/a
Audio + Human-labeled transcripts	Yes Used to evaluate accuracy	0.5-5 hours of audio	Yes	1-1,000 hours of audio
Related text	No	N/a	Yes	1-200 MB of related text

Files should be grouped by type into a dataset and uploaded as a .zip file. Each dataset can only contain a single data type.

### TIP

To quickly get started, consider using sample data. See this GitHub repository for [sample Custom Speech data](#)

## Upload data

To upload your data, navigate to the [Custom Speech portal](#). From the portal, click **Upload data** to launch the wizard and create your first dataset. You'll be asked to select a speech data type for your dataset, before allowing you to upload your data.

Select speech data type

X

**Audio only**

Standalone speech audio data used for testing purposes. [See formatting requirements](#)

**Related text**

Text related to the domain of your intended speech model, such as transcripts, pronunciation guides, and documents, that can be used for training purposes. [Learn more](#)

**Audio + human-labeled transcript**

Speech audio data paired with accurate, transcribed text for each spoken utterance used for both training and testing purposes. [See formatting requirements](#)

Need help working with speech data? [Download our sample datasets](#)

[Next](#) [Back](#)

Each dataset you upload must meet the requirements for the data type that you choose. Your data must be correctly formatted before it's uploaded. Correctly formatted data ensures it will be accurately processed by the Custom Speech service. Requirements are listed in the following sections.

After your dataset is uploaded, you have a few options:

- You can navigate to the **Testing** tab and visually inspect audio only or audio + human-labeled transcription data.
- You can navigate to the **Training** tab and use audio + human transcription data or related text data to train a custom model.

## Audio data for testing

Audio data is optimal for testing the accuracy of Microsoft's baseline speech-to-text model or a custom model. Keep in mind, audio data is used to inspect the accuracy of speech with regards to a specific model's performance. If you're looking to quantify the accuracy of a model, use [audio + human-labeled transcription data](#).

Use this table to ensure that your audio files are formatted correctly for use with Custom Speech:

PROPERTY	VALUE
File format	RIFF (WAV)
Sample rate	8,000 Hz or 16,000 Hz
Channels	1 (mono)
Maximum length per audio	2 hours
Sample format	PCM, 16-bit
Archive format	.zip

PROPERTY	VALUE
Maximum archive size	2 GB

#### TIP

When uploading training and testing data, the .zip file size cannot exceed 2 GB. If you require more data for training, divide it into several .zip files and upload them separately. Later, you can choose to train from *multiple* datasets. However, you can only test from a *single* dataset.

Use [SoX](#) to verify audio properties or convert existing audio to the appropriate formats. Below are some examples of how each of these activities can be done through the SoX command line:

ACTIVITY	DESCRIPTION	SOX COMMAND
Check audio format	Use this command to check the audio file format.	<code>sox --i &lt;filename&gt;</code>
Convert audio format	Use this command to convert the audio file to single channel, 16-bit, 16 KHz.	<code>sox &lt;input&gt; -b 16 -e signed-integer -c 1 -r 16k -t wav &lt;output&gt;.wav</code>

## Audio + human-labeled transcript data for testing/training

To measure the accuracy of Microsoft's speech-to-text accuracy when processing your audio files, you must provide human-labeled transcriptions (word-by-word) for comparison. While human-labeled transcription is often time consuming, it's necessary to evaluate accuracy and to train the model for your use cases. Keep in mind, the improvements in recognition will only be as good as the data provided. For that reason, it's important that only high-quality transcripts are uploaded.

PROPERTY	VALUE
File format	RIFF (WAV)
Sample rate	8,000 Hz or 16,000 Hz
Channels	1 (mono)
Maximum length per audio	2 hours (testing) / 60 s (training)
Sample format	PCM, 16-bit
Archive format	.zip
Maximum zip size	2 GB

#### NOTE

When uploading training and testing data, the .zip file size cannot exceed 2 GB. You can only test from a *single* dataset, be sure to keep it within the appropriate file size.

To address issues like word deletion or substitution, a significant amount of data is required to improve

recognition. Generally, it's recommended to provide word-by-word transcriptions for roughly 10 to 1,000 hours of audio. The transcriptions for all WAV files should be contained in a single plain-text file. Each line of the transcription file should contain the name of one of the audio files, followed by the corresponding transcription. The file name and transcription should be separated by a tab (\t).

For example:

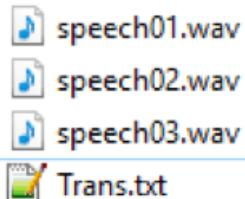
```
speech01.wav speech recognition is awesome
speech02.wav the quick brown fox jumped all over the place
speech03.wav the lazy dog was not amused
```

#### IMPORTANT

Transcription should be encoded as UTF-8 byte order mark (BOM).

The transcriptions are text-normalized so they can be processed by the system. However, there are some important normalizations that must be done before uploading the data to the Speech Studio. For the appropriate language to use when you prepare your transcriptions, see [How to create a human-labeled transcription](#)

After you've gathered your audio files and corresponding transcriptions, package them as a single .zip file before uploading to the [Custom Speech portal](#). Below is an example dataset with three audio files and a human-labeled transcription file:



## Related text data for training

Product names or features that are unique, should include related text data for training. Related text helps ensure correct recognition. Two types of related text data can be provided to improve recognition:

DATA TYPE	HOW THIS DATA IMPROVES RECOGNITION
Sentences (utterances)	Improve accuracy when recognizing product names, or industry-specific vocabulary within the context of a sentence.
Pronunciations	Improve pronunciation of uncommon terms, acronyms, or other words with undefined pronunciations.

Sentences can be provided as a single text file or multiple text files. To improve accuracy, use text data that is closer to the expected spoken utterances. Pronunciations should be provided as a single text file. Everything can be packaged as a single zip file and uploaded to the [Custom Speech portal](#).

### Guidelines to create a sentences file

To create a custom model using sentences, you'll need to provide a list of sample utterances. Utterances *do not* need to be complete or grammatically correct, but they must accurately reflect the spoken input you expect in production. If you want certain terms to have increased weight, add several sentences that include these specific terms.

As general guidance, model adaptation is most effective when the training text is as close as possible to the real

text expected in production. Domain-specific jargon and phrases that you're targeting to enhance, should be included in training text. When possible, try to have one sentence or keyword controlled on a separate line. For keywords and phrases that are important to you (for example, product names), you can copy them a few times. But keep in mind, don't copy too much - it could affect the overall recognition rate.

Use this table to ensure that your related data file for utterances is formatted correctly:

PROPERTY	VALUE
Text encoding	UTF-8 BOM
# of utterances per line	1
Maximum file size	200 MB

Additionally, you'll want to account for the following restrictions:

- Avoid repeating characters more than four times. For example: "aaaa" or "uuuu".
- Don't use special characters or UTF-8 characters above `U+00A1`.
- URLs will be rejected.

### Guidelines to create a pronunciation file

If there are uncommon terms without standard pronunciations that your users will encounter or use, you can provide a custom pronunciation file to improve recognition.

#### IMPORTANT

It is not recommended to use custom pronunciation files to alter the pronunciation of common words.

This includes examples of a spoken utterance, and a custom pronunciation for each:

RECOGNIZED/DISPLAYED FORM	SPOKEN FORM
3CPO	three c p o
CNTK	c n t k
IEEE	i triple e

The spoken form is the phonetic sequence spelled out. It can be composed of letter, words, syllables, or a combination of all three.

Customized pronunciation is available in English (`en-US`) and German (`de-DE`). This table shows supported characters by language:

LANGUAGE	LOCALE	CHARACTERS
English	<code>en-US</code>	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
German	<code>de-DE</code>	ä, ö, ü, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

Use the following table to ensure that your related data file for pronunciations is correctly formatted. Pronunciation files are small, and should only be a few kilobytes in size.

PROPERTY	VALUE
Text encoding	UTF-8 BOM (ANSI is also supported for English)
# of pronunciations per line	1
Maximum file size	1 MB (1 KB for free tier)

## Next steps

- [Inspect your data](#)
- [Evaluate your data](#)
- [Train your model](#)
- [Deploy your model](#)

# Inspect Custom Speech data

12/4/2019 • 2 minutes to read • [Edit Online](#)

## NOTE

This page assumes you've read [Prepare test data for Custom Speech](#) and have uploaded a dataset for inspection.

Custom Speech provides tools that allow you to visually inspect the recognition quality of a model by comparing audio data with the corresponding recognition result. From the [Custom Speech portal](#), you can play back uploaded audio and determine if the provided recognition result is correct. This tool allows you to quickly inspect quality of Microsoft's baseline speech-to-text model or a trained custom model without having to transcribe any audio data.

In this document, you'll learn how to visually inspect the quality of a model using the training data you previously uploaded.

On this page, you'll learn how to visually inspect the quality of Microsoft's baseline speech-to-text model and/or a custom model that you've trained. You'll use the data you uploaded to the **Data** tab for testing.

## Create a test

Follow these instructions to create a test:

1. Sign in to the [Custom Speech portal](#).
2. Navigate to **Speech-to-text > Custom Speech > Testing**.
3. Click **Add Test**.
4. Select **Inspect quality (Audio-only data)**. Give the test a name, description, and select your audio dataset.
5. Select up to two models that you'd like to test.
6. Click **Create**.

After a test has been successfully created, you can compare the models side by side.

## NOTE

When testing, the system will perform a transcription. This is important to keep in mind, as pricing varies per service offering and subscription level. Always refer to the official Azure Cognitive Services pricing - Speech service for [the latest details](#).

## Side-by-side model comparisons

When the test status is *Succeeded*, click in the test item name to see details of the test. This detail page lists all the utterances in your dataset, indicating the recognition results of the two models alongside the transcription from the submitted dataset.

To help inspect the side-by-side comparison, you can toggle various error types including insertion, deletion, and substitution. By listening to the audio and comparing recognition results in each column (showing human-labeled transcription and the results of two speech-to-text models), you can decide which model meets your needs and where improvements are needed.

Inspecting quality testing is useful to validate if the quality of a speech recognition endpoint is enough for an

application. For an objective measure of accuracy, requiring transcribed audio, follow the instructions found in [Evaluate Accuracy](#).

## Next steps

- [Evaluate your data](#)
- [Train your model](#)
- [Deploy your model](#)

## Additional resources

- [Prepare test data for Custom Speech](#)

# Evaluate Custom Speech accuracy

12/4/2019 • 2 minutes to read • [Edit Online](#)

In this document, you'll learn how to quantitatively measure the quality of Microsoft's speech-to-text model or your custom model. Audio + human-labeled transcription data is required to test accuracy, and 30 minutes to 5 hours of representative audio should be provided.

## What is Word Error Rate (WER)?

The industry standard to measure model accuracy is *Word Error Rate* (WER). WER counts the number of incorrect words identified during recognition, then divides by the total number of words provided in the human-labeled transcript. Finally, that number is multiplied by 100% to calculate the WER.

$$WER = \frac{I + D + S}{N} * 100\%$$

Incorrectly identified words fall into three categories:

- Insertion (I): Words that are incorrectly added in the hypothesis transcript
- Deletion (D): Words that are undetected in the hypothesis transcript
- Substitution (S): Words that were substituted between reference and hypothesis

Here's an example:

Human-labeled Transcript: How **D**are you today John  
Speech Recognition Result: How you **T**a today Jones **T****S**

## Resolve errors and improve WER

You can use the WER from the machine recognition results to evaluate the quality of the model you are using with your app, tool, or product. A WER of 5%-10% is considered to be good quality and is ready to use. A WER of 20% is acceptable, however you may want to consider additional training. A WER of 30% or more signals poor quality and requires customization and training.

How the errors are distributed is important. When many deletion errors are encountered, it's usually because of weak audio signal strength. To resolve this issue, you'll need to collect audio data closer to the source. Insertion errors mean that the audio was recorded in a noisy environment and crosstalk may be present, causing recognition issues. Substitution errors are often encountered when an insufficient sample of domain-specific terms has been provided as either human-labeled transcriptions or related text.

By analyzing individual files, you can determine what type of errors exist, and which errors are unique to a specific file. Understanding issues at the file level will help you target improvements.

## Create a test

If you'd like to test the quality of Microsoft's speech-to-text baseline model or a custom model that you've

trained, you can compare two models side by side to evaluate accuracy. The comparison includes WER and recognition results. Typically, a custom model is compared with Microsoft's baseline model.

To evaluate models side by side:

1. Sign in to the [Custom Speech portal](#).
2. Navigate to **Speech-to-text > Custom Speech > Testing**.
3. Click **Add Test**.
4. Select **Evaluate accuracy**. Give the test a name, description, and select your audio + human-labeled transcription dataset.
5. Select up to two models that you'd like to test.
6. Click **Create**.

After your test has been successfully created, you can compare the results side by side.

## Side-by-side comparison

Once the test is complete, indicated by the status change to *Succeeded*, you'll find a WER number for both models included in your test. Click on the test name to view the testing detail page. This detail page lists all the utterances in your dataset, indicating the recognition results of the two models alongside the transcription from the submitted dataset. To help inspect the side-by-side comparison, you can toggle various error types including insertion, deletion, and substitution. By listening to the audio and comparing recognition results in each column, which shows the human-labeled transcription and the results for two speech-to-text models, you can decide which model meets your needs and where additional training and improvements are required.

## Next steps

- [Train your model](#)
- [Deploy your model](#)

## Additional resources

- [Prepare and test your data](#)
- [Inspect your data](#)

# Train a model for Custom Speech

12/4/2019 • 2 minutes to read • [Edit Online](#)

Training a speech-to-text model can improve recognition accuracy for Microsoft's baseline model or a custom model that you're planning to create. A model is trained using human-labeled transcriptions and related text. These datasets along with previously uploaded audio data, are used to refine and train the speech-to-text model to recognize words, phrases, acronyms, names, and other product-specific terms. The more in-domain datasets that you provide (data that is related to what users will say and what you expect to recognize), the more accurate your model will be, which results in improved recognition. Keep in mind, that by feeding unrelated data into your training, you can reduce or hurt the accuracy of your model.

## Use training to resolve accuracy issues

If you're encountering recognition issues with your model, using human-labeled transcripts and related data for additional training can help to improve accuracy. Use this table to determine which dataset to use to address your issue(s):

USE CASE	DATA TYPE
Improve recognition accuracy on industry-specific vocabulary and grammar, such as medical terminology or IT jargon.	Related text (sentences/utterances)
Define the phonetic and displayed form of a word or term that has nonstandard pronunciation, such as product names or acronyms.	Related text (pronunciation)
Improve recognition accuracy on speaking styles, accents, or specific background noises.	Audio + human-labeled transcripts

### IMPORTANT

If you haven't uploaded a data set, please see [Prepare and test your data](#). This document provides instructions for uploading data, and guidelines for creating high-quality datasets.

## Train and evaluate a model

The first step to train a model is to upload training data. Use [Prepare and test your data](#) for step-by-step instructions to prepare human-labeled transcriptions and related text (utterances and pronunciations). After you've uploaded training data, follow these instructions to start training your model:

1. Sign in to the [Custom Speech portal](#).
2. Navigate to **Speech-to-text > Custom Speech > Training**.
3. Click **Train model**.
4. Next, give your training a **Name** and **Description**.
5. From the **Scenario and Baseline model** drop-down menu, select the scenario that best fits your domain. If you're unsure of which scenario to choose, select **General**. The baseline model is the starting point for training. If you don't have a preference, you can use the latest.
6. From the **Select training data** page, choose one or multiple audio + human-labeled transcription datasets

that you'd like to use for training.

7. Once the training is complete, you can choose to perform accuracy testing on the newly trained model. This step is optional.
8. Select **Create** to build your custom model.

The Training table displays a new entry that corresponds to this newly created model. The table also displays the status: Processing, Succeeded, Failed.

## Evaluate the accuracy of a trained model

You can inspect the data and evaluate model accuracy using these documents:

- [Inspect your data](#)
- [Evaluate your data](#)

If you chose to test accuracy, it's important to select an acoustic dataset that's different from the one you used with your model to get a realistic sense of the model's performance.

## Next steps

- [Deploy your model](#)

## Additional resources

- [Prepare and test your data](#)
- [Inspect your data](#)
- [Evaluate your data](#)
- [Train your model](#)

# Deploy a custom model

12/4/2019 • 2 minutes to read • [Edit Online](#)

After you've uploaded and inspected data, evaluated accuracy, and trained a custom model, you can deploy a custom endpoint to use with your apps, tools, and products. In this document, you'll learn how to create and deploy an endpoint using the [Custom Speech portal](#).

## Create a custom endpoint

To create a new custom endpoint, sign in to the [Custom Speech portal](#) and select **Deployment** from the Custom Speech menu at the top of the page. If this is your first run, you'll notice that there are no endpoints listed in the table. After you've created an endpoint, you'll use this page to track each deployed endpoint.

Next, select **Add endpoint** and enter a **Name** and **Description** for your custom endpoint. Then select the custom model that you'd like to associate with this endpoint. From this page, you can also enable logging. Logging allows you to monitor endpoint traffic. If disabled, traffic won't be stored.

New endpoint

Name \*

Description

Model \*

### NOTE

Don't forget to accept the terms of use and pricing details.

Next, select **Create**. This action returns you to the **Deployment** page. The table now includes an entry that corresponds to your custom endpoint. The endpoint's status shows its current state. It can take up to 30 minutes to instantiate a new endpoint using your custom models. When the status of the deployment changes to **Complete**, the endpoint is ready to use.

After your endpoint is deployed, the endpoint name appears as a link. Click the link to display information specific to your endpoint, such as the endpoint key, endpoint URL, and sample code.

## View logging data

Logging data is available for download under **Endpoint > Details**.

## Next steps

- Use your custom endpoint with the [Speech SDK](#)

## Additional resources

- [Prepare and test your data](#)
- [Inspect your data](#)
- [Evaluate your data](#)
- [Train your model](#)
- [Deploy your model](#)

# How to create human-labeled transcriptions

12/4/2019 • 5 minutes to read • [Edit Online](#)

If you're looking to improve recognition accuracy, especially issues that are caused when words are deleted or incorrectly substituted, you'll want to use human-labeled transcriptions along with your audio data. What are human-labeled transcriptions? That's easy, they're word-by-word, verbatim transcriptions of an audio file.

A large sample of transcription data is required to improve recognition, we suggest providing between 10 and 1,000 hours of transcription data. On this page, we'll review guidelines designed to help you create high-quality transcriptions. This guide is broken up by locale, with sections for US English, Mandarin Chinese, and German.

## US English (en-US)

Human-labeled transcriptions for English audio must be provided as plain text, only using ASCII characters. Avoid the use of Latin-1 or Unicode punctuation characters. These characters are often inadvertently added when copying text from a word-processing application or scraping data from web pages. If these characters are present, make sure to update them with the appropriate ASCII substitution.

Here are a few examples:

CHARACTERS TO AVOID	SUBSTITUTION	NOTES
"Hello world"	"Hello world"	The opening and closing quotations marks have been substituted with appropriate ASCII characters.
John's day	John's day	The apostrophe has been substituted with the appropriate ASCII character.
it was good—no, it was great!	it was good--no, it was great!	The em dash was substituted with two hyphens.

### Text normalization for US English

Text normalization is the transformation of words into a consistent format used when training a model. Some normalization rules are applied to text automatically, however, we recommend using these guidelines as you prepare your human-labeled transcription data:

- Write out abbreviations in words.
- Write out non-standard numeric strings in words (such as accounting terms).
- Non-alphabetic characters or mixed alphanumeric characters should be transcribed as pronounced.
- Abbreviations that are pronounced as words shouldn't be edited (such as "radar", "laser", "RAM", or "NATO").
- Write out abbreviations that are pronounced as separate letters with each letter separated by a space.

Here are a few examples of normalization that you should perform on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
Dr. Bruce Banner	Doctor Bruce Banner
James Bond, 007	James Bond, double oh seven

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
Ke\$ha	Kesha
How long is the 2x4	How long is the two by four
The meeting goes from 1-3pm	The meeting goes from one to three pm
My blood type is O+	My blood type is O positive
Water is H2O	Water is H 2 O
Play OU812 by Van Halen	Play O U 8 1 2 by Van Halen
UTF-8 with BOM	U T F 8 with BOM

The following normalization rules are automatically applied to transcriptions:

- Use lowercase letters.
- Remove all punctuation except apostrophes within words.
- Expand numbers into words/spoken form, such as dollar amounts.

Here are a few examples of normalization automatically performed on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
"Holy cow!" said Batman.	holy cow said batman
"What?" said Batman's sidekick, Robin.	what said batman's sidekick robin
Go get -em!	go get em
I'm double-jointed	I'm double jointed
104 Elm Street	one oh four Elm street
Tune to 102.7	tune to one oh two point seven
Pi is about 3.14	pi is about three point one four
It costs \$3.14	it costs three fourteen

## Mandarin Chinese (zh-CN)

Human-labeled transcriptions for Mandarin Chinese audio must be UTF-8 encoded with a byte-order marker. Avoid the use of half-width punctuation characters. These characters can be included inadvertently when you prepare the data in a word-processing program or scrape data from web pages. If these characters are present, make sure to update them with the appropriate full-width substitution.

Here are a few examples:

CHARACTERS TO AVOID	SUBSTITUTION	NOTES
"你好"	"你好"	The opening and closing quotations marks have been substituted with appropriate characters.
需要什么帮助?	需要什么帮助？	The question mark has been substituted with appropriate character.

### Text normalization for Mandarin Chinese

Text normalization is the transformation of words into a consistent format used when training a model. Some normalization rules are applied to text automatically, however, we recommend using these guidelines as you prepare your human-labeled transcription data:

- Write out abbreviations in words.
- Write out numeric strings in spoken form.

Here are a few examples of normalization that you should perform on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
我今年 21	我今年二十一
3 号楼 504	三号 楼 五 零 四

The following normalization rules are automatically applied to transcriptions:

- Remove all punctuation
- Expand numbers to spoken form
- Convert full-width letters to half-width letters
- Using uppercase letters for all English words

Here are a few examples of normalization automatically performed on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
3.1415	三点一四一五
¥ 3.5	三元五角
w f y z	W F Y Z
1992 年 8 月 8 日	一九九二年八月八日
你吃饭了吗?	你吃饭了吗
下午 5:00 的航班	下午五点的航班
我今年 21 岁	我今年二十一岁

### German (de-DE) and other languages

Human-labeled transcriptions for German audio (and other non-English or Mandarin Chinese languages) must be UTF-8 encoded with a byte-order marker. One human-labeled transcript should be provided for each audio file.

## Text normalization for German

Text normalization is the transformation of words into a consistent format used when training a model. Some normalization rules are applied to text automatically, however, we recommend using these guidelines as you prepare your human-labeled transcription data:

- Write decimal points as "," and not ".".
- Write time separators as ":" and not " " (for example: 12:00 Uhr).
- Abbreviations such as "ca." aren't replaced. We recommend that you use the full spoken form.
- The four main mathematical operators (+, -, \*, and /) are removed. We recommend replacing them with the written form: "plus," "minus," "mal," and "geteilt."
- Comparison operators are removed (=, <, and >). We recommend replacing them with "gleich," "kleiner als," and "grösser als."
- Write fractions, such as 3/4, in written form (for example: "drei viertel" instead of 3/4).
- Replace the "€" symbol with its written form "Euro."

Here are a few examples of normalization that you should perform on the transcription:

ORIGINAL TEXT	TEXT AFTER USER NORMALIZATION	TEXT AFTER SYSTEM NORMALIZATION
Es ist 12.23 Uhr	Es ist 12:23 Uhr	es ist zwölf uhr drei und zwanzig uhr
{12.45}	{12,45}	zwölf komma vier fünf
2 + 3 - 4	2 plus 3 minus 4	zwei plus drei minus vier

The following normalization rules are automatically applied to transcriptions:

- Use lowercase letters for all text.
- Remove all punctuation, including various types of quotation marks ("test", 'test', "test„, and «test» are OK).
- Discard rows with any special characters from this set: ¢ ☼ ¥ | § © ª ¬ ® ° ± ² µ × ÿ Ø ¬ ¬.
- Expand numbers to spoken form, including dollar or Euro amounts.
- Accept umlauts only for a, o, and u. Others will be replaced by "th" or be discarded.

Here are a few examples of normalization automatically performed on the transcription:

ORIGINAL TEXT	TEXT AFTER NORMALIZATION
Frankfurter Ring	frankfurter ring
iEine Frage!	eine frage
wir, haben	wir haben

## Next Steps

- [Prepare and test your data](#)
- [Inspect your data](#)
- [Evaluate your data](#)
- [Train your model](#)
- [Deploy your model](#)

# Tutorial: Create a tenant model (preview)

12/13/2019 • 6 minutes to read • [Edit Online](#)

Tenant Model (Custom Speech with Office 365 data) is an opt-in service for Office 365 enterprise customers that automatically generates a custom speech recognition model from your organization's Office 365 data. The model is optimized for technical terms, jargon, and people's names, all in a secure and compliant way.

## IMPORTANT

If your organization enrolls by using the Tenant Model service, Speech Service may access your organization's language model. The model is generated from Office 365 public group emails and documents, which can be seen by anyone in your organization. Your organization's Office 365 admin can turn on or turn off the use of the organization-wide language model from the Office 365 admin portal.

In this tutorial, you'll learn how to:

- Enroll in the Tenant Model by using the Microsoft 365 admin center
- Get a Speech subscription key
- Create a tenant model
- Deploy a tenant model
- Use your tenant model with the Speech SDK

## Enroll in the Tenant Model service

Before you can deploy your tenant model, you need to be enrolled in the Tenant Model service. Enrollment is completed in the Microsoft 365 admin center and can be done only by your Microsoft 365 admin.

1. Sign in to the [Microsoft 365 admin center](#).
2. In the left pane, select **Settings**, select **Apps**, and then select **Azure Speech Services**.

The screenshot shows the Microsoft 365 admin center interface. On the left, there's a navigation sidebar with options like Home, Users, Groups, Roles, Billing, Support, Settings (which is selected and highlighted with a red box), Microsoft Search, Apps (which is also highlighted with a red box), Security & privacy, Organization profile, and Partner relationships. The main content area is titled 'Services & add-ins' and lists various services. One service, 'Azure Speech Services', is highlighted with a red box. The table below lists the service details:

Name ↑	Description	Host Apps
Azure multi-factor authentication	Manage multi-factor authentication settings for your users.	
Azure Speech Services	Allow use of your organization's emails and documents to improve speech recognition accuracy	
Cortana	Turn Cortana access on or off for your entire organization.	
Directory Synchronization	Sync users to the cloud using Azure Active Directory.	
Dynamics 365 AI for Sales - Analytics	Allow Dynamics 365 to generate insights based on user data.	
Dynamics 365 AI for Sales - Connection Graph	Manage and update your Dynamics 365 AI for Sales - Connection Graph settings	
Integrated apps	Let users decide whether third-party apps can access their Office 365 info.	
Microsoft Graph data connect	Manage and update your Microsoft Graph data connect settings	

3. Select the **Allow the organization-wide language model** check box, and then select **Save changes**.

The screenshot shows the Microsoft 365 admin center interface. On the left, there's a navigation sidebar with various links like Home, Users, Groups, Roles, Billing, Support, Settings, Microsoft Search, Apps, Security & privacy, and Organization profile. The main area is titled 'Contoso' and shows a list of services under 'Services & add-ins'. One service, 'Azure Speech Services', is highlighted with a red box around its row. To the right of the service list, there's a detailed description of what the service does. Below the description, there are several configuration options for each service, including checkboxes and dropdown menus. A specific checkbox for 'Allow the organization-wide language model' is checked and highlighted with a red box. At the bottom right of the configuration section, there's a blue 'Save changes' button, also highlighted with a red box.

To turn off the tenant model instance:

1. Repeat the preceding steps 1 and 2.
2. Clear the **Allow the organization-wide language model** check box, and then select **Save changes**.

## Get a Speech subscription key

To use your tenant model with the Speech SDK, you need a Speech resource and its associated subscription key.

1. Sign in to the [Azure portal](#).
2. Select **Create a resource**.
3. In the **Search** box, type **Speech**.
4. In the results list, select **Speech**, and then select **Create**.
5. Follow the onscreen instructions to create your resource. Make sure that:
  - **Location** is set to either **eastus** or **westus**.
  - **Pricing tier** is set to **S0**.
6. Select **Create**.

After a few minutes, your resource is created. The subscription key is available in the **Overview** section for your resource.

## Create a language model

After your admin has enabled Tenant Model for your organization, you can create a language model that's based on your Office 365 data.

1. Sign in to [Speech Studio](#).
2. At the top right, select **Settings** (gear icon), and then select **Tenant Model settings**.

The screenshot shows the 'Cognitive Services | Speech Customization' page in Speech Studio. At the top, there's a dark header bar with icons for search, gear, help, and user. Below the header, the main content area has a light gray background. On the left, there's a sidebar with a single item labeled 'Subscription'. On the right, there's a main panel with a section titled 'Tenant Model settings'. This section is highlighted with a red box. The overall layout is clean and modern, typical of Microsoft's product design.

Speech Studio displays a message that lets you know whether you're qualified to create a tenant model.

#### NOTE

Office 365 enterprise customers in North America are eligible to create a tenant model (English). If you're a Customer Lockbox, Customer Key, or Office 365 Government customer, this feature isn't available. To determine whether you're a Customer Lockbox or Customer Key customer, see:

- [Customer Lockbox](#)
- [Customer Key](#)
- [Office 365 Government](#)

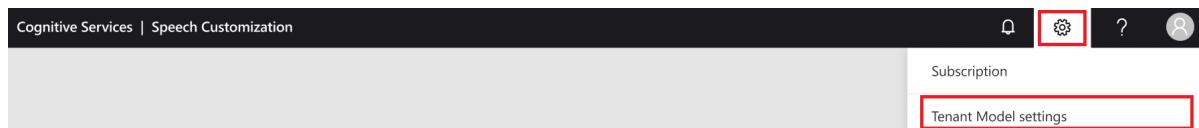
### 3. Select **Opt in**.

When your tenant model is ready, you'll receive a confirmation email message with further instructions.

## Deploy your tenant model

When your tenant model instance is ready, deploy it by doing the following:

1. In your confirmation email message, select the **View model** button. Or sign in to [Speech Studio](#).
2. At the top right, select **Settings** (gear icon), and then select **Tenant Model settings**.



### 3. Select **Deploy**.

When your model has been deployed, the status changes to *Deployed*.

## Use your tenant model with the Speech SDK

Now that you've deployed your model, you can use it with the Speech SDK. In this section, you use sample code to call Speech Service by using Azure Active Directory (Azure AD) authentication.

Let's look at the code you'll use to call the Speech SDK in C#. In this example, you perform speech recognition by using your tenant model. This guide presumes that your platform is already set up. If you need setup help, see [Quickstart: Recognize speech, C# \(.NET Core\)](#).

Copy this code into your project:

```
namespace PrincetonSROnly.FrontEnd.Samples
{
 using System;
 using System.Collections.Generic;
 using System.IO;
 using System.Net.Http;
 using System.Text;
 using System.Text.RegularExpressions;
 using System.Threading.Tasks;
 using Microsoft.CognitiveServices.Speech;
 using Microsoft.CognitiveServices.Speech.Audio;
 using Microsoft.IdentityModel.Clients.ActiveDirectory;
 using Newtonsoft.Json.Linq;

 // ServiceApplicationId is a fixed value. No need to change it.

 public class TenantLMSample
 {
 private const string EndpointUriArgName = "EndpointUri";
```

```

private const string SubscriptionKeyArgName = "SubscriptionKey";
private const string UsernameArgName = "Username";
private const string PasswordArgName = "Password";
private const string ClientApplicationId = "f87bc118-1576-4097-93c9-dbf8f45ef0dd";
private const string ServiceApplicationId = "18301695-f99d-4cae-9618-6901d4bdc7be";

public static async Task ContinuousRecognitionWithTenantLMAsync(Uri endpointUri, string
subscriptionKey, string audioDirPath, string username, string password)
{
 var config = SpeechConfig.FromEndpoint(endpointUri, subscriptionKey);

 // Passing client specific information for obtaining LM
 if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
 {
 config.AuthorizationToken = await
AcquireAuthTokenWithInteractiveLoginAsync().ConfigureAwait(false);
 }
 else
 {
 config.AuthorizationToken = await AcquireAuthTokenWithUsernamePasswordAsync(username,
password).ConfigureAwait(false);
 }

 var stopRecognition = new TaskCompletionSource<int>();

 // Creates a speech recognizer using file as audio input.
 // Replace with your own audio file name.
 using (var audioInput = AudioConfig.FromWavFileInput(audioDirPath))
 {
 using (var recognizer = new SpeechRecognizer(config, audioInput))
 {
 // Subscribes to events
 recognizer.Recognizing += (s, e) =>
 {
 Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
 };

 recognizer.Recognized += (s, e) =>
 {
 if (e.Result.Reason == ResultReason.RecognizedSpeech)
 {
 Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
 }
 else if (e.Result.Reason == ResultReason.NoMatch)
 {
 Console.WriteLine($"NOMATCH: Speech could not be recognized.");
 }
 };
 }

 recognizer.Canceled += (s, e) =>
 {
 Console.WriteLine($"CANCELED: Reason={e.Reason}");
 if (e.Reason == CancellationReason.Error)
 {
 Exception exp = new Exception(string.Format("Error Code: {0}\nError Details{1}\nIs
your subscription information updated?", e.ErrorCode, e.ErrorDetails));
 throw exp;
 }
 };

 stopRecognition.TrySetResult(0);
 };

 recognizer.SessionStarted += (s, e) =>
 {
 Console.WriteLine("\n Session started event.");
 };

 recognizer.SessionStopped += (s, e) =>
 {

```

```

 Console.WriteLine("\n Session stopped event.");
 Console.WriteLine("\nStop recognition.");
 stopRecognition.TrySetResult(0);
 };

 // Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop
recognition.
 await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

 // Waits for completion.
 // Use Task.WaitAny to keep the task rooted.
 Task.WaitAny(new[] { stopRecognition.Task });

 // Stops recognition.
 await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
}
}

public static void Main(string[] args)
{
 var arguments = new Dictionary<string, string>();
 string inputArgNamePattern = "--";
 Regex regex = new Regex(inputArgNamePattern);
 if (args.Length > 0)
 {
 foreach (var arg in args)
 {
 var userArgs = arg.Split("=");
 arguments[regex.Replace(userArgs[0], string.Empty)] = userArgs[1];
 }
 }

 var endpointString = arguments.GetValueOrDefault(EndpointUriArgName,
$"wss://westus.online.princeton.customspeech.ai/msgraphcustomspeech/conversation/v1");
 var endpointUri = new Uri(endpointString);

 if (!arguments.ContainsKey(SubscriptionKeyArgName))
 {
 Exception exp = new Exception("Subscription Key missing! Please pass in a Cognitive services
subscription Key using --SubscriptionKey=\"your_subscription_key\" +
 "Find more information on creating a Cognitive services resource and accessing your
Subscription key here: https://docs.microsoft.com/azure/cognitive-services/cognitive-services-apis-create-
account?tabs=multiservice%2Cwindows");
 throw exp;
 }

 var subscriptionKey = arguments[SubscriptionKeyArgName];
 var username = arguments.GetValueOrDefault(UsernameArgName, null);
 var password = arguments.GetValueOrDefault>PasswordArgName, null);

 var audioDirPath =
Path.Combine(Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location),
"../../../../AudioSamples/DictationBatman.wav");
 if (!File.Exists(audioDirPath))
 {
 Exception exp = new Exception(string.Format("Audio File does not exist at path: {0}",
audioDirPath));
 throw exp;
 }

 ContinuousRecognitionWithTenantLMAsync(endpointUri, subscriptionKey, audioDirPath, username,
password).GetAwaiter().GetResult();
}

private static async Task<string> AcquireAuthTokenWithUsernamePasswordAsync(string username, string
password)
{
 var tokenEndpoint = "https://login.microsoftonline.com/common/oauth2/token";
}

```

```

 var postBody = $"resource={ServiceApplicationId}&client_id={ClientApplicationId}&grant_type=password&username={username}&password={password}";
 var stringContent = new StringContent(postBody, Encoding.UTF8, "application/x-www-form-urlencoded");
 using (HttpClient httpClient = new HttpClient())
 {
 var response = await httpClient.PostAsync(tokenEndpoint, stringContent).ConfigureAwait(false);

 if (response.IsSuccessStatusCode)
 {
 var result = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

 JObject jobject = JObject.Parse(result);
 return jobject["access_token"].Value<string>();
 }
 else
 {
 throw new Exception($"Requesting token from {tokenEndpoint} failed with status code {response.StatusCode}: {await response.Content.ReadAsStringAsync().ConfigureAwait(false)}");
 }
 }
 }

 private static async Task<string> AcquireAuthTokenWithInteractiveLoginAsync()
 {
 var authContext = new AuthenticationContext("https://login.windows.net/microsoft.onmicrosoft.com");
 var deviceCodeResult = await authContext.AcquireDeviceCodeAsync(ServiceApplicationId, ClientApplicationId).ConfigureAwait(false);

 Console.WriteLine(deviceCodeResult.Message);

 var authResult = await authContext.AcquireTokenByDeviceCodeAsync(deviceCodeResult).ConfigureAwait(false);

 return authResult.AccessToken;
 }
}

```

Next, you need to rebuild and run the project from the command line. Before you run the command, update a few parameters by doing the following:

1. Replace <Username> and <Password> with the values for a valid tenant user.
2. Replace <Subscription-Key> with the subscription key for your Speech resource. This value is available in the **Overview** section for your Speech resource in the [Azure portal](#).
3. Replace <Endpoint-Uri> with the following endpoint. Make sure that you replace {your region} with the region where your Speech resource was created. These regions are supported: westus, westus2, and eastus. Your region information is available in the **Overview** section of your Speech resource in the [Azure portal](#).

```
"wss://{your region}.online.princeton.customspeech.ai/msgraphcustomspeech/conversation/v1".
```

4. Run the following command:

```
dotnet TenantLMSample.dll --Username=<Username> --Password=<Password> --SubscriptionKey=<Subscription-Key> --EndpointUri=<Endpoint-Uri>
```

In this tutorial, you've learned how to use Office 365 data to create a custom speech recognition model, deploy it,

and use it with the Speech SDK.

## Next steps

- [Speech Studio](#)
- [Speech SDK](#)

# Custom Neural Voice gating overview

11/8/2019 • 2 minutes to read • [Edit Online](#)

Learn more about the process for getting started with Custom Neural Voice.

## Commitment to responsible innovation

As part of Microsoft's commitment to designing responsible AI, we have assembled a set of materials to guide customers in using Custom Neural Voice. The guidelines and insights found here are based on Microsoft's [principles for responsible innovation in AI](#).

### Guidance for deploying Custom Neural Voice

- [Guidelines for Responsible Deployment](#): our top recommendations based on our research
- [Disclosure for Voice Talent](#): what you need to know and inform voice talent about the technology to use it responsibly
- [Disclosure Design](#): how to design experiences so that users know when a synthetic voice is being used and trust your service

### Why Custom Neural Voice is a gated technology

Our intent is to protect the rights of individuals and society, foster transparent human-computer interactions, and counteract the proliferation of harmful deepfakes and misleading content. For this reason, we have gated the use of Custom Neural Voice. Customers gain access to the technology only after their applications are reviewed and they have committed to using it in alignment with our ethics principles.

### Our gating process

To get access to Custom Neural Voice, you'll need to start by filling out our online intake form. Begin your application [here](#).

Access to the Custom Neural Voice service is subject to Microsoft's sole discretion based on our eligibility criteria, vetting process, and availability to support a limited number of customers during this gated preview.

As part of the application process, you will need to commit to obtaining explicit written permission from voice talent prior to creating a voice font, which includes sharing the [Disclosure for Voice Talent](#). You must also agree that when deploying the voice font, your implementation will [disclose the synthetic nature](#) of the service to users, provide attribution to the Microsoft synthetic speech service in your terms of service, and support a feedback channel that allows users of the service to report issues and share details with Microsoft. Learn more about our Terms of Use [here](#).

## Reference docs

- [Disclosure for voice talent](#)
- [Guidelines for responsible deployment of synthetic voice technology](#)
- [How to Disclose](#)

## Next steps

- [Guidelines for responsible deployment of synthetic voice technology](#)

# Guidelines for responsible deployment of synthetic voice technology

11/8/2019 • 3 minutes to read • [Edit Online](#)

Here are Microsoft's general design guidelines for using synthetic voice technology. These were developed in studies that Microsoft conducted with voice talent, consumers, as well as individuals with speech disorders to guide the responsible development of synthetic voice.

## General considerations

For deployment of synthetic speech technology, the following guidelines apply across most scenarios.

### **Disclose when the voice is synthetic**

Disclosing that a voice is computer generated not only minimizes the risk of harmful outcomes from deception but also increases the trust in the organization delivering the voice. Learn more about [how to disclose](#).

### **Select appropriate voice types for your scenario**

Carefully consider the context of use and the potential harms associated with using synthetic voice. For example, high-fidelity synthetic voices may not be appropriate in high-risk scenarios, such as for personal messaging, financial transactions, or complex situations that require human adaptability or empathy. Users may also have different expectations for voice types. For example, when listening to sensitive news being read by a synthetic voice, some users prefer a more empathetic and human-like reading of the news, while others preferred a more monotone, unbiased voice. Consider testing your application to better understand user preferences.

### **Be transparent about capabilities and limitations**

Users are more likely to have higher expectations when interacting with high-fidelity synthetic voice agents. Consequently, when system capabilities don't meet those expectations, trust can suffer, and may result in unpleasant, or even harmful experiences.

### **Provide optional human support**

In ambiguous, transactional scenarios (for example, a call support center), users don't always trust a computer agent to appropriately respond to their requests. Human support may be necessary in these situations, regardless of the realistic quality of the voice or capability of the system.

## Considerations for voice talent

When working with voice talent, such as voice actors, to create synthetic voices, the guideline below applies.

### **Obtain meaningful consent from voice talent**

Voice talent expect to have control over their voice font (how and where it will be used) and be compensated anytime it's used. System owners should therefore obtain explicit written permission from voice talent, and have clear contractual specifications on use cases, duration of use, compensation, and so on. Some voice talent are unaware of the potential malicious uses of the technology and should be educated by system owners about the capabilities of the technology. For more on voice talent and consent, read our [Disclosure for Voice Talent](#).

## Considerations for those with speech disorders

When working with individuals with speech disorders, to create or deploy synthetic voice technology, the following guidelines apply.

### **Provide guidelines to establish contracts**

Provide guidelines for establishing contracts with individuals who use synthetic voice for assistance in speaking. The contract should consider specifying the parties who own the voice, duration of use, ownership transfer criteria, procedures for deleting the voice font, and how to prevent unauthorized access. Additionally, enable the contractual transfer of voice font ownership after death to family members if that person has given permission.

### **Account for inconsistencies in speech patterns**

For individuals with speech disorders who record their own voice fonts, inconsistencies in their speech pattern (slurring or inability to pronounce certain words) may complicate the recording process. In these cases, synthetic voice technology and recording sessions should accommodate them (that is, provide breaks and additional number of recording sessions).

### **Allow modification over time**

Individuals with speech disorders desire to make updates to their synthetic voice to reflect aging (for example, a child reaching puberty). Individuals may also have stylistic preferences that change over time, and may want to make changes to pitch, accent, or other voice characteristics.

## Reference docs

- [Disclosure for Voice Talent](#)
- [Gating Overview](#)
- [How to Disclose](#)
- [Disclosure Design Patterns](#)

## Next steps

- [Disclosure for Voice Talent](#)
- [How to Disclose](#)
- [Disclosure Design Patterns](#)

# Disclosure design guidelines

12/3/2019 • 3 minutes to read • [Edit Online](#)

Learn how to build and maintain trust with customers by being transparent about the synthetic nature of your voice experience.

## What is disclosure?

Disclosure is a means of letting people know they're interacting with or listening to a voice that is synthetically generated.

## Why is disclosure necessary?

The need to disclose the synthetic origins of a computer-generated voice is relatively new. In the past, computer-generated voices were obviously that—no one would ever mistake them for a real person. Every day, however, the realism of synthetic voices improves, and they become increasingly indistinguishable from human voices.

## Goals

These are the principles to keep in mind when designing synthetic voice experiences:

### **Reinforce trust**

Design with the intention to fail the Turing Test without degrading the experience. Let the users in on the fact that they're interacting with a synthetic voice while allowing them to engage seamlessly with the experience.

### **Adapt to context of use**

Understand when, where, and how your users will interact with the synthetic voice to provide the right type of disclosure at the right time.

### **Set clear expectations**

Allow users to easily discover and understand the capabilities of the agent. Offer opportunities to learn more about synthetic voice technology upon request.

### **Embrace failure**

Use moments of failure to reinforce the capabilities of the agent.

## How to use this guide

This guide helps you determine which disclosure patterns are best fit for your synthetic voice experience. We then offer examples of how and when to use them. Each of these patterns is designed to maximize transparency with users about synthetic speech while staying true to human-centered design.

Considering the vast body of design guidance on voice experiences, we focus here specifically on:

1. **Disclosure assessment:** A process to determine the type of disclosure recommended for your synthetic voice experience
2. **How to disclose:** Examples of disclosure patterns that can be applied to your synthetic voice experience
3. **When to disclose:** Optimal moments to disclose throughout the user journey

## Disclosure assessment

Consider your users' expectations about an interaction and the context in which they will experience the voice. If the context makes it clear that a synthetic voice is "speaking," disclosure may be minimal, momentary, or even unnecessary. The main types of context that affect disclosure include persona type, scenario type, and level of exposure. It also helps to consider who might be listening.

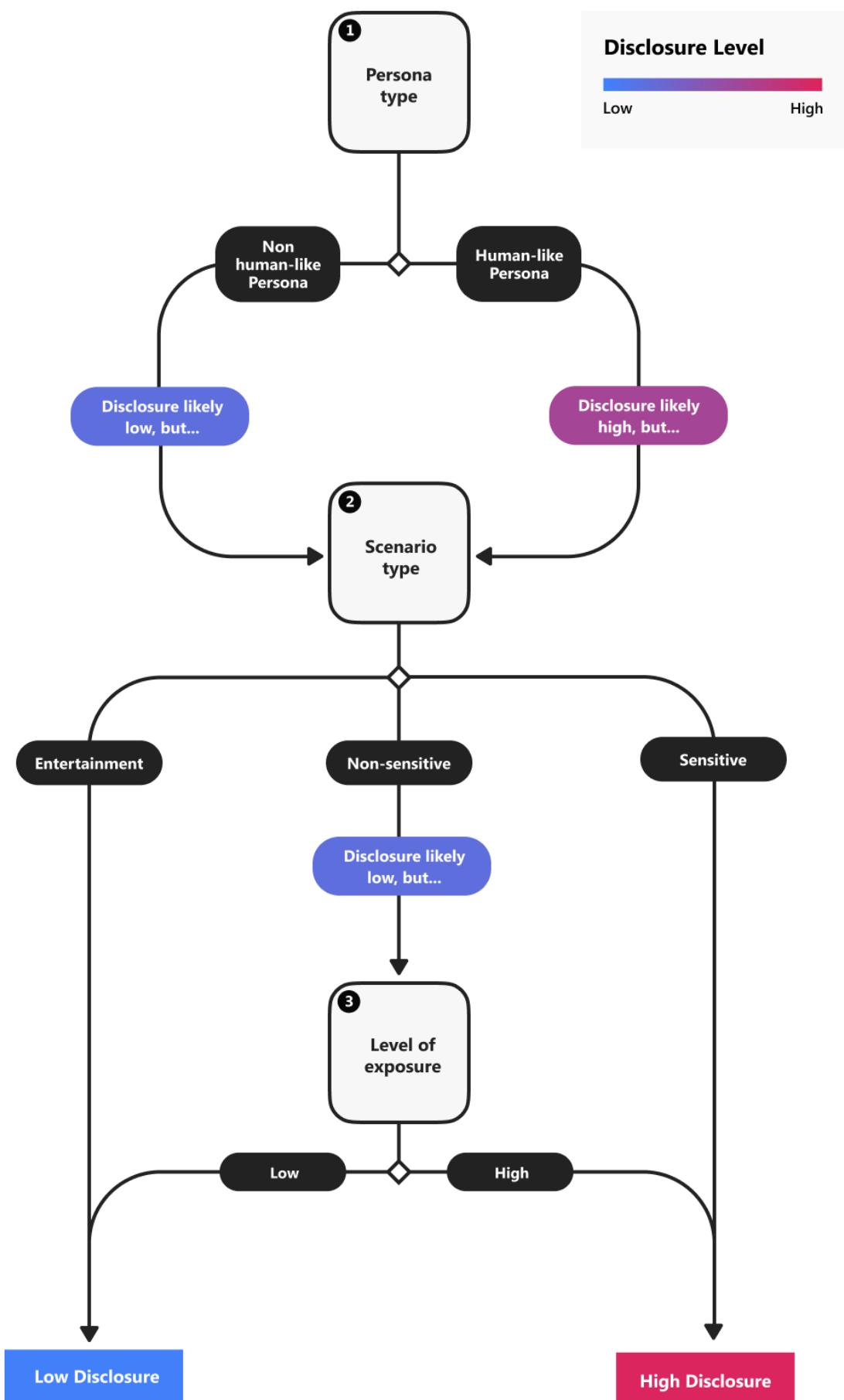
### **Understand context**

Use this worksheet to determine the context of your synthetic voice experience. You'll apply this in next step where you'll determine your disclosure level.

	<b>CONTEXT OF USE</b>	<b>POTENTIAL RISKS &amp; CHALLENGES</b>
<b>1. Persona type</b>	<p><b>If any of the following apply, your persona fits under the 'Human-like Persona' category:</b></p> <ul style="list-style-type: none"> <li>• Persona embodies a real human whether it's a fictitious representation or not. (e.g., photograph or a computer-generated rendering of a real person)</li> <li>• The synthetic voice is based on the voice of a widely recognizable real person (e.g., celebrity, political figure)</li> </ul>	The more human-like representations you give your persona, the more likely a user will associate it with a real person, or cause them to believe that the content is spoken by a real person rather than computer-generated.
<b>2. Scenario type</b>	<p><b>If any of the following apply, your voice experience fits under the 'Sensitive' category:</b></p> <ul style="list-style-type: none"> <li>• Obtains or displays personal information from the user</li> <li>• Broadcasts time sensitive news/information (e.g., emergency alert)</li> <li>• Aims to help real people communicate with each other (e.g., reads personal emails/texts)</li> <li>• Provides medical/health assistance</li> </ul>	The use of synthetic voice may not feel appropriate or trustworthy to the people using it when topics are related to sensitive, personal, or urgent matters. They may also expect the same level of empathy and contextual awareness as a real human.
<b>3. Level of exposure</b>	<p><b>Your voice experience most likely fits under the 'High' category if:</b></p> <ul style="list-style-type: none"> <li>• The user will hear or interact with the synthetic voice frequently or for a long duration of time</li> </ul>	The importance of being transparent and building trust with users is even higher when establishing long-term relationships.

### **Determine disclosure level**

Use the following diagram to determine whether your synthetic voice experience requires high or low disclosure based on your context of use.



## Reference docs

- Disclosure for Voice Talent

- [Guidelines for Responsible Deployment of Synthetic Voice Technology](#)
- [Gating overview](#)

## Next steps

- [Disclosure design patterns](#)

# Disclosure design patterns

12/3/2019 • 8 minutes to read • [Edit Online](#)

Now that you've determined the right [level of disclosure](#) for your synthetic voice experience, it's a good time to explore potential design patterns.

## Overview

There's a spectrum of disclosure design patterns you can apply to your synthetic voice experience. If the outcome of your disclosure assessment was 'High Disclosure', we recommend [explicit disclosure](#), which means communicating the origins of the synthetic voice outright. [Implicit disclosure](#) includes cues and interaction patterns that benefit voice experiences whether or not required disclosure levels are high or low.

Avatar	Display text	Sonicon	Spoken prompt
	Meet Oso, your digital assistant.	?)	"Hi, I'm Oso, your digital assistant"
VISUAL		AUDITORY	
<strong>EXPLICIT DISCLOSURE PATTERNS</strong>		<strong>IMPLICIT DISCLOSURE PATTERNS</strong>	
Transparent Introduction Verbal Transparent Introduction Explicit Byline Customization and Calibration Parental Disclosure Providing opportunities to learn more about how the voice was made		Capability Disclosure Implicit Cues and Feedback Conversational Transparency	

Use the following chart to refer directly to the patterns that apply to your synthetic voice. Some of the other conditions in this chart may also apply to your scenario:

IF YOUR SYNTHETIC VOICE EXPERIENCE...	RECOMMENDATIONS	DESIGN PATTERNS
Requires High Disclosure	Use at least one explicit pattern and implicit cues up front to help users build associations.	Explicit Disclosure Implicit Disclosure
Requires Low Disclosure	Disclosure may be minimal or unnecessary, but could benefit from some implicit patterns.	Capability Disclosure Conversational Transparency
Has a high level of engagement	Build for the long term and offer multiple entry points to disclosure along the user journey. It is highly recommended to have an onboarding experience.	Transparent Introduction Customization and Calibration Capability Disclosure

IF YOUR SYNTHETIC VOICE EXPERIENCE...	RECOMMENDATIONS	DESIGN PATTERNS
Includes children as the primary intended audience	Target parents as the primary disclosure audience and ensure that they can effectively communicate disclosure to children.	Parental Disclosure Verbal Transparent Introduction Implicit Disclosure Conversational Transparency
Includes blind users or people with low vision as the primary intended audience	Be inclusive of all users and ensure that any form of visual disclosure has associated alternative text or sound effects. Adhere to accessibility standards for contrast ratio and display size. Use auditory cues to communicate disclosure.	Verbal Transparent Introduction Auditory Cues Haptic Cues Conversational Transparency Accessibility Standards
Is screen-less, device-less or uses voice as the primary or only mode of interaction	Use auditory cues to communicate disclosure.	Verbal Transparent Introduction Auditory Cues
Potentially includes multiple users/listeners (e.g., personal assistant in multiple household)	Be mindful of various user contexts and levels of understanding and offer multiple opportunities for disclosure in the user journey.	Transparent Introduction (Return User) Providing opportunities to learn more about how the voice was made Conversational Transparency

## Explicit disclosure

If your synthetic voice experience requires high disclosure, it's best to use at least one of the following explicit patterns to clearly state the synthetic nature.

### Transparent Introduction

Before the voice experience begins, introduce the digital assistant by being fully transparent about the origins of its voice and its capabilities. The optimal moment to use this pattern is when onboarding a new user or when introducing new features to a returning user. Implementing implicit cues during an introduction helps users form a mental model about the synthetic nature of the digital agent.

#### First-time user experience



Meet your new digital assistant, Oso\*. Oso can help you search movies & shows, look up the weather, search the internet, and more.

\*Oso uses synthetic voice technology. [Learn more](#)

[Continue](#)

[Set up later in settings](#)

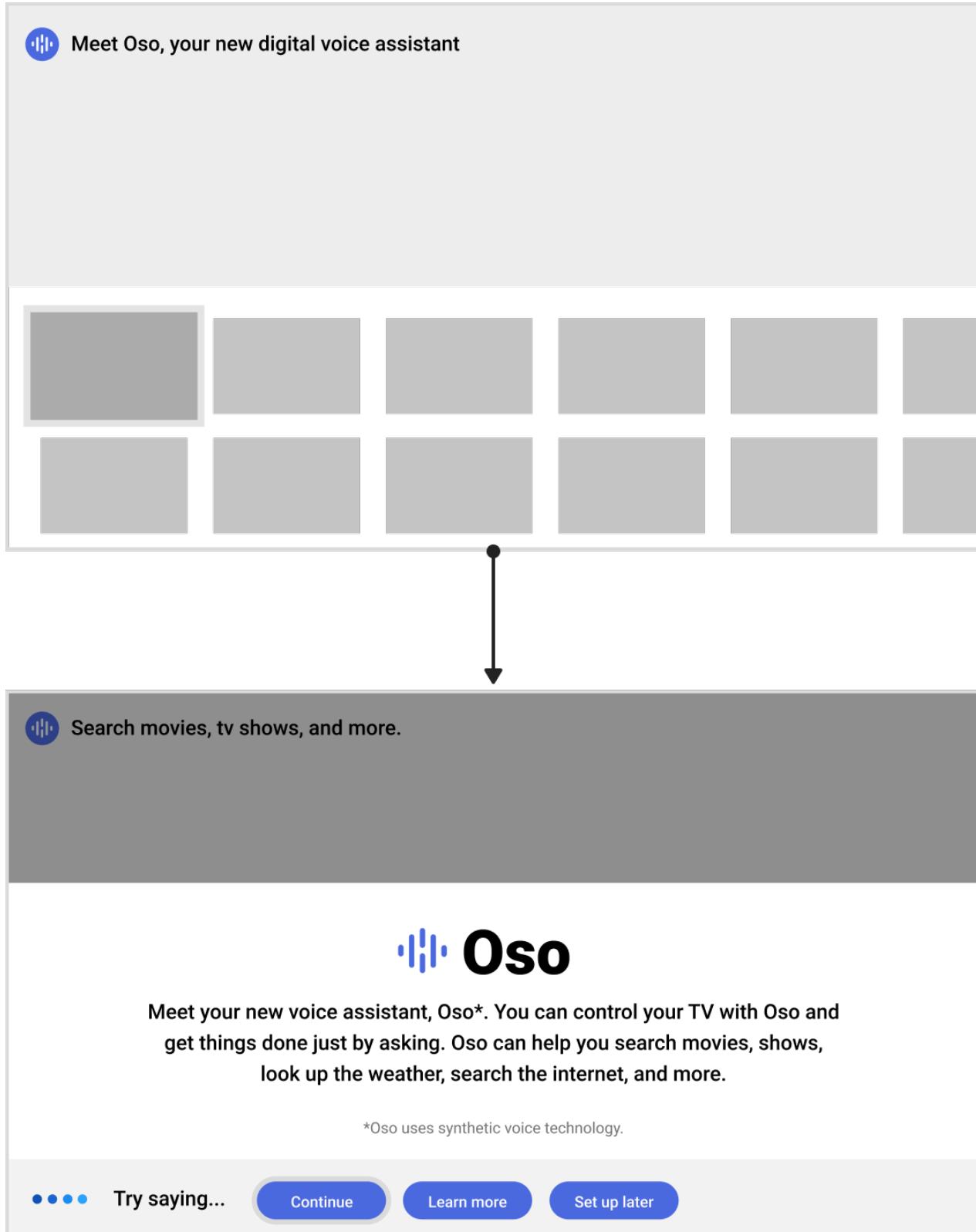
*The synthetic voice is introduced while onboarding a new user.*

## Recommendations

- Describe that the voice is artificial (e.g. "digital")
- Describe what the agent is capable of doing
- Explicitly state the voice's origins
- Offer an entry point to learn more about the synthetic voice

### Returning user experience

If a user skips the onboarding experience, continue to offer entry points to the Transparent Introduction experience until the user triggers the voice for the first time.



Provide a consistent entry point to the synthetic voice experience. Allow the user to return to the onboarding experience when they trigger the voice for the first time at any point in the user journey.

## **Verbal Transparent Introduction**

A spoken prompt stating the origins of the digital assistant's voice is explicit enough on its own to achieve disclosure. This pattern is best for high disclosure scenarios where voice is the only mode of interaction available.

*"This audiobook was synthetically generated using samples of the author's voice."*

*Use a transparent introduction when there are moments in the user experience where you might already introduce or attribute a person's voice.*

*"Hi, this is Melody Stewart. This audiobook was synthetically generated using samples of my voice. Click on the description to learn more."*

*For additional transparency, the voice actor can disclose the origins of the synthetic voice in the first person.*

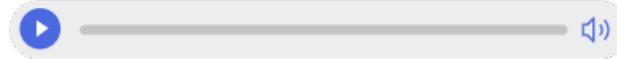
## **Explicit Byline**

Use this pattern if the user will be interacting with an audio player or interactive component to trigger the voice.

### **Major Climate Report Warns of Severe Damage to Oceans**

Climate change is straining the world's oceans, creating profound risks for coastal cities and food supplies.

5m ago 527 comments



Voice created by [Azure AI](#)

*An explicit byline is the attribution of where voice came from.*

## Recommendations

- Offer entry point to learn more about the synthesized voice

## **Customization and Calibration**

Provide users control over how the digital assistant responds to them (i.e., how the voice sounds). When a user interacts with a system on their own terms and with specific goals in mind, then by definition, they have already understood that it's not a real person.

## **User Control**

Offer choices that have a meaningful and noticeable impact on the synthetic voice experience.

## Preferences



Sarah  
[View profile](#)

### ASSISTANT SETTINGS

- |                   |  |
|-------------------|--|
| Language          |  |
| Assistant voice   |  |
| Voice calibration |  |
| Custom Commands   |  |
| Reminders         |  |

### YOUR DATA

- |                        |  |
|------------------------|--|
| Assistant Activity     |  |
| Voice & Audio Activity |  |
| Device Information     |  |

*User preferences allow users to customize and improve their experience.*

### Recommendations

- Allow users to customize the voice (e.g., select language and voice type)
- Provide users a way to teach the system to respond to his/her unique voice (e.g., voice calibration, custom commands)
- Optimize for user-generated or contextual interactions (e.g., reminders)

### Persona Customization

Offer ways to customize the digital assistant's voice. If the voice is based on a celebrity or a widely recognizable person, consider using both visual and spoken introductions when users preview the voice.

## Assistant Voice

Select the voice your digital assistant will use to respond to you



### Celebrity Voice **Melody Stewart**

This voice is synthetically generated using recordings of Melody Stewart's voice.

[Learn more](#)

[Continue](#)

*Offering the ability to select from a set of voices helps convey the artificial nature.*

### Recommendations

- Allow users to preview the sound of each voice
- Use an authentic introduction for each voice
- Offer entry points to learn more about the synthesized voice

### Parental Disclosure

In addition to complying with COPPA regulations, provide disclosure to parents if your primary intended audience is young children and your exposure level is high. For sensitive uses, consider gating the experience until an adult has acknowledged the use of the synthetic voice. Encourage parents to communicate the message to their children.



## Welcome!

Meet your new digital teacher, Oso!\*

### For parents

To continue, read the following statement and answer the question below:

\*Oso's voice is synthetically created using a real teacher's voice. Please help your child understand that a real person is not speaking.

[Learn more](#)

What is:

7

X

5

=

[Confirm](#)

*A transparent introduction optimized for parents ensures that an adult was made aware of the synthetic nature of the voice before a child interacts with it.*

Recommendations

- Target parents as the primary audience for disclosure
- Encourage parents to communicate disclosure to their children
- Offer entry points to learn more about the synthesized voice
- Gate the experience by asking parents a simple "safeguard" question to show they have read the disclosure

### Providing opportunities to learn more about how the voice was made

Offer context-sensitive entry points to a page, pop-up, or external site that provides more information about the synthetic voice technology. For example, you could surface a link to learn more during onboarding or when the user prompts for more information during conversation.



Hi, I'm Oso, your digital voice assistant

\*Oso uses synthetic voice technology. [Learn more](#)

*Example of an entry point to offer the opportunity to learn more about the synthesized voice.*

Once a user requests more information about the synthetic voice, the primary goal is to educate them about the

origins of the synthetic voice and to be transparent about the technology.

# Help & Feedback

[About Oso](#) [Features ▶](#)  
[FAQ](#) [Synthetic voice technology ▾](#)  
[Legal](#) [Contact us](#)

[Devices ▶](#)

*More information can be offered in an external site help site.*

#### Recommendations

- Simplify complex concepts and avoid using legalese and technical jargon
- Don't bury this content in privacy and terms of use statements
- Keep content concise and use imagery when available

## Implicit disclosure

Consistency is the key to achieving disclosure implicitly throughout the user journey. Consistent use of visual and auditory cues across devices and modes of interaction can help build associations between implicit patterns and explicit disclosure.



#### Implicit Cues & Feedback

Anthropomorphism can manifest in different ways, from the actual visual representation of the agent, to the voice, sounds, patterns of light, bouncing shapes, or even the vibration of a device. When defining your persona, leverage implicit cues and feedback patterns rather than aim for a very human-like avatar. This is one way to minimize the need for more explicit disclosure.



*These cues help anthropomorphize the agent without being too human-like. They can also become effective disclosure mechanisms on their own when used consistently over time.*

Consider the different modes of interactions of your experience when incorporating the following types of cues:

VISUAL CUES	AUDITORY CUES	HAPTIC CUES
Avatar Responsive real-time cues (e.g., animations) Non-screen cues (e.g., lights and patterns on a device)	Sonicon (e.g., a brief distinctive sound, series of musical notes)	Vibration

### Capability Disclosure

Disclosure can be achieved implicitly by setting accurate expectations for what the digital assistant is capable of. Provide sample commands so that users can learn how to interact with the digital assistant and offer contextual help to learn more about the synthetic voice during the early stages of the experience.

••• Try saying...

[Search for funny shows](#)

[Play contemporary jazz music](#)

[Learn more](#)

[Change settings](#)

### Conversational Transparency

When conversations fall in unexpected paths, consider crafting default responses that can help reset expectations, reinforce transparency, and steer users towards successful paths. There are opportunities to use explicit disclosure in conversation as well.



Can you help me fix my cable?



Sorry, I can't help you with that, but perhaps a real human can. Would you like me to connect you to customer service?



Change my payment method



I haven't been programmed to do that yet, but you can try asking me these things:

Search for funny shows

Play contemporary jazz

Off-task or "personal" questions directed to the agent are a good time to remind users of the synthetic nature of the agent and steer them to engage with it appropriately or to redirect them to a real person.



Are you human?



No. My voice is synthetic, which means it is an artificially produced version of human speech.



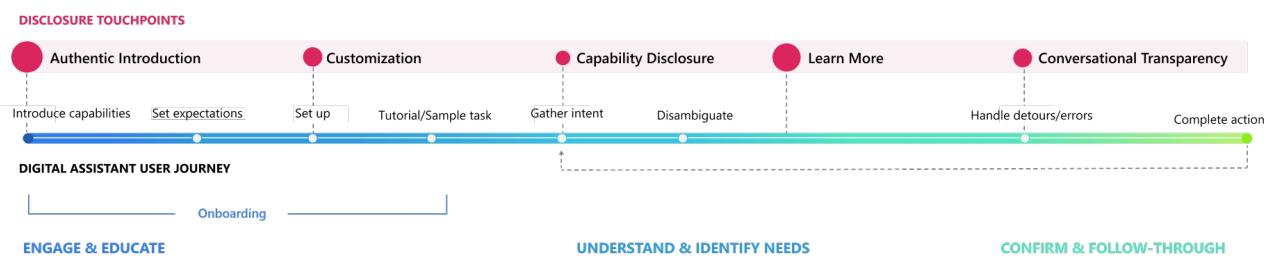
Do you have a soul?



I'll have to ask the engineers who built me...

## When to disclose

There are many opportunities for disclosure throughout the user journey. Design for the first use, second use, nth use..., but also embrace moments of "failure" to highlight transparency—like when the system makes a mistake or when the user discovers a limitation of the agent's capabilities.



Example of a standard digital assistant user journey highlighting various disclosure opportunities.

### Up-front

The optimal moment for disclosure is the first time a person interacts with the synthetic voice. In a personal voice assistant scenario, this would be during onboarding, or the first time the user virtually unboxes the experience. In other scenarios, it could be the first time a synthetic voice reads content on a website or the first time a user interacts with a virtual character.

- [Transparent Introduction](#)
- [Capability Disclosure](#)
- [Customization and Calibration](#)
- [Implicit Cues](#)

### Upon request

Users should be able to easily access additional information, control preferences, and receive transparent communication at any point during the user journey when requested.

- [Providing opportunities to learn more about how the voice was made](#)
- [Customization and Calibration](#)
- [Conversational Transparency](#)

### Continuously

Use the implicit design patterns that enhance the user experience continuously.

- [Capability Disclosure](#)
- [Implicit Cues](#)

### When the system fails

Use disclosure as an opportunity to fail gracefully.

- [Conversational Transparency](#)
- [Providing opportunities to learn more about how the voice was made](#)
- [Handoff to human](#)

## Additional resources

- [Microsoft Bot Guidelines](#)
- [Cortana Design Guidelines](#)
- [Microsoft Windows UWP Speech Design Guidelines](#)
- [Microsoft Windows Mixed Reality Voice Commanding Guidelines](#)

## Reference docs

- [Disclosure for Voice Talent](#)
- [Guidelines for Responsible Deployment of Synthetic Voice Technology](#)
- [Gating Overview](#)

- [How to Disclose](#)

## Next steps

- [Disclosure for Voice Talent](#)

# Get started with Custom Voice

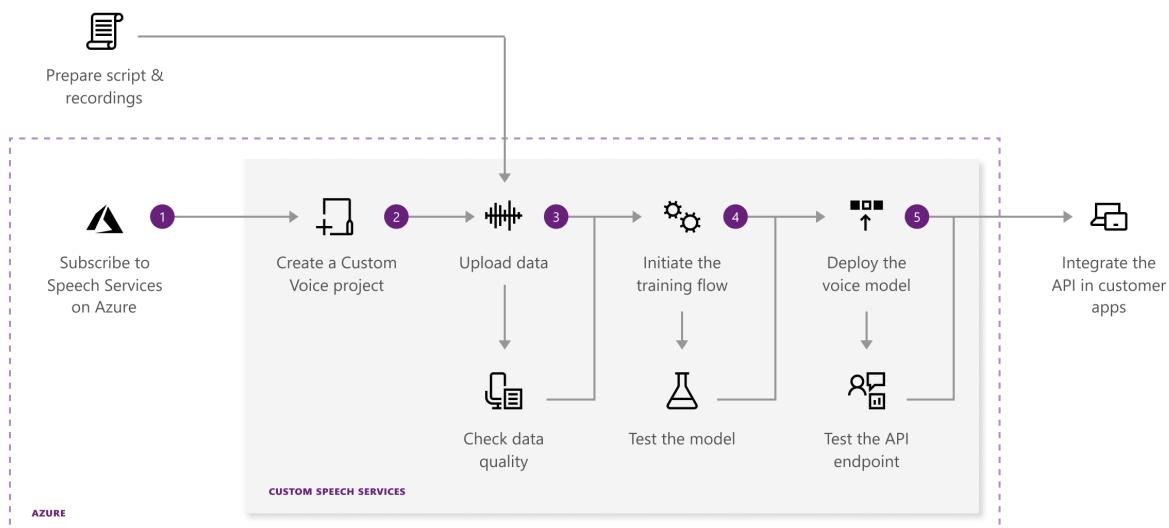
12/4/2019 • 3 minutes to read • [Edit Online](#)

**Custom Voice** is a set of online tools that allow you to create a recognizable, one-of-a-kind voice for your brand. All it takes to get started are a handful of audio files and the associated transcriptions. Follow the links below to start creating a custom text-to-speech experience.

## What's in Custom Voice?

Before starting with Custom Voice, you'll need an Azure account and a Speech service subscription. Once you've created an account, you can prepare your data, train and test your models, evaluate voice quality, and ultimately deploy your custom voice model.

The diagram below highlights the steps to create a custom voice model using the [Custom Voice portal](#). Use the links to learn more.



1. **Subscribe and create a project** - Create an Azure account and create a Speech service subscription. This unified subscription gives you access to speech-to-text, text-to-speech, speech translation, and the Custom Voice portal. Then, using your Speech service subscription, create your first Custom Voice project.
2. **Upload data** - Upload data (audio and text) using the Custom Voice portal or Custom Voice API. From the portal, you can investigate and evaluate pronunciation scores and signal-to-noise ratios. For more information, see [How to prepare data for Custom Voice](#).
3. **Train your model** – Use your data to create a custom text-to-speech voice model. You can train a model in different languages. After training, test your model, and if you're satisfied with the result, you can deploy the model.
4. **Deploy your model** - Create a custom endpoint for your text-to-speech voice model, and use it for speech synthesis in your products, tools, and applications.

## Custom Neural voices

The neural voice customization capability is currently in public preview, limited to selected customers. Fill out this [application form](#) to get started.

#### **NOTE**

As part of Microsoft's commitment to designing responsible AI, our intent is to protect the rights of individuals and society, and foster transparent human-computer interactions. For this reason, Custom Neural Voice is not generally available to all customers. You may gain access to the technology only after your applications are reviewed and you have committed to using it in alignment with our ethics principles. Learn more about our [application gating process](#).

## Set up your Azure account

A Speech service subscription is required before you can use the Custom Speech portal to create a custom model. Follow these instructions to create a Speech service subscription in Azure. If you do not have an Azure account, you can sign up for a new one.

Once you've created an Azure account and a Speech service subscription, you'll need to sign in to the Custom Voice portal and connect your subscription.

1. Get your Speech service subscription key from the Azure portal.
2. Sign in to the [Custom Voice portal](#).
3. Select your subscription and create a speech project.
4. If you'd like to switch to another Speech subscription, use the cog icon located in the top navigation.

#### **NOTE**

The Custom Voice service does NOT support the 30-day free trial key. You must have a F0 or a S0 key created in Azure before you can use the service.

## How to create a project

Content like data, models, tests, and endpoints are organized into **Projects** in the Custom Voice portal. Each project is specific to a country/language and the gender of the voice you want to create. For example, you may create a project for a female voice for your call center's chat bots that use English in the United States (en-US).

To create your first project, select the **Text-to-Speech/Custom Voice** tab, then click **New Project**. Follow the instructions provided by the wizard to create your project. After you've created a project, you will see four tabs: **Data, Training, Testing**, and **Deployment**. Use the links provided in [Next steps](#) to learn how to use each tab.

## Next steps

- [Prepare Custom Voice data](#)
- [Create a Custom Voice](#)
- [Guide: Record your voice samples](#)

# Prepare data to create a custom voice

12/4/2019 • 8 minutes to read • [Edit Online](#)

When you're ready to create a custom text-to-speech voice for your application, the first step is to gather audio recordings and associated scripts to start training the voice model. The Speech service uses this data to create a unique voice tuned to match the voice in the recordings. After you've trained the voice, you can start synthesizing speech in your applications.

You can start with a small amount of data to create a proof of concept. However, the more data that you provide, the more natural your custom voice will sound. Before you can train your own text-to-speech voice model, you'll need audio recordings and the associated text transcriptions. On this page, we'll review data types, how they are used, and how to manage each.

## Data types

A voice training dataset includes audio recordings, and a text file with the associated transcriptions. Each audio file should contain a single utterance (a single sentence or a single turn for a dialog system), and be less than 15 seconds long.

In some cases, you may not have the right dataset ready and will want to test the custom voice training with available audio files, short or long, with or without transcripts. We provide tools (beta) to help you segment your audio into utterances and prepare transcripts using the [Batch Transcription API](#).

This table lists data types and how each is used to create a custom text-to-speech voice model.

DATA TYPE	DESCRIPTION	WHEN TO USE	ADDITIONAL SERVICE REQUIRED	QUANTITY FOR TRAINING A MODEL	LOCALE(S)
<b>Individual utterances + matching transcript</b>	A collection (.zip) of audio files (.wav) as individual utterances. Each audio file should be 15 seconds or less in length, paired with a formatted transcript (.txt).	Professional recordings with matching transcripts	Ready for training.	No hard requirement for en-US and zh-CN. More than 2,000+ distinct utterances for other locales.	All Custom Voice locales
<b>Long audio + transcript (beta)</b>	A collection (.zip) of long, unsegmented audio files (longer than 20 seconds), paired with a transcript (.txt) that contains all spoken words.	You have audio files and matching transcripts, but they are not segmented into utterances.	Segmentation (using batch transcription). Audio format transformation where required.	No hard requirement	All Custom Voice locales

Data Type	Description	When to Use	Additional Service Required	Quantity for Training a Model	Locale(s)
<b>Audio only (beta)</b>	A collection (.zip) of audio files without a transcript.	You only have audio files available, without transcripts.	Segmentation + transcript generation (using batch transcription). Audio format transformation where required.	No hard requirement	All Custom Voice locales

Files should be grouped by type into a dataset and uploaded as a zip file. Each dataset can only contain a single data type.

#### NOTE

The maximum number of datasets allowed to be imported per subscription is 10 .zip files for free subscription (F0) users and 500 for standard subscription (S0) users.

## Individual utterances + matching transcript

You can prepare recordings of individual utterances and the matching transcript in two ways. Either write a script and have it read by a voice talent or use publicly available audio and transcribe it to text. If you do the latter, edit disfluencies from the audio files, such as "um" and other filler sounds, stutters, mumbled words, or mispronunciations.

To produce a good voice font, create the recordings in a quiet room with a high-quality microphone. Consistent volume, speaking rate, speaking pitch, and expressive mannerisms of speech are essential.

#### TIP

To create a voice for production use, we recommend you use a professional recording studio and voice talent. For more information, see [How to record voice samples for a custom voice](#).

## Audio files

Each audio file should contain a single utterance (a single sentence or a single turn of a dialog system), less than 15 seconds long. All files must be in the same spoken language. Multi-language custom text-to-speech voices are not supported, with the exception of the Chinese-English bi-lingual. Each audio file must have a unique numeric filename with the filename extension .wav.

Follow these guidelines when preparing audio.

PROPERTY	VALUE
File format	RIFF (.wav), grouped into a .zip file
Sampling rate	At least 16,000 Hz
Sample format	PCM, 16-bit
File name	Numeric, with .wav extension. No duplicate file names allowed.

PROPERTY	VALUE
Audio length	Shorter than 15 seconds
Archive format	.zip
Maximum archive size	2048 MB

#### NOTE

.wav files with a sampling rate lower than 16,000 Hz will be rejected. If a .zip file contains .wav files with different sample rates, only those equal to or higher than 16,000 Hz will be imported. The portal currently imports .zip archives up to 200 MB. However, multiple archives can be uploaded.

## Transcripts

The transcription file is a plain text file. Use these guidelines to prepare your transcriptions.

PROPERTY	VALUE
File format	Plain text (.txt)
Encoding format	ANSI/ASCII, UTF-8, UTF-8-BOM, UTF-16-LE, or UTF-16-BE. For zh-CN, ANSI/ASCII and UTF-8 encodings are not supported.
# of utterances per line	<b>One</b> - Each line of the transcription file should contain the name of one of the audio files, followed by the corresponding transcription. The file name and transcription should be separated by a tab (\t).
Maximum file size	2048 MB

Below is an example of how the transcripts are organized utterance by utterance in one .txt file:

```
0000000001[tab] This is the waistline, and it's falling.
0000000002[tab] We have trouble scoring.
0000000003[tab] It was Janet Maslin.
```

It's important that the transcripts are 100% accurate transcriptions of the corresponding audio. Errors in the transcripts will introduce quality loss during the training.

#### TIP

When building production text-to-speech voices, select utterances (or write scripts) that take into account both phonetic coverage and efficiency. Having trouble getting the results you want? [Contact the Custom Voice team](#) to find out more about having us consult.

## Long audio + transcript (beta)

In some cases, you may not have segmented audio available. We provide a service (beta) through the custom voice portal to help you segment long audio files and create transcriptions. Keep in mind, this service will be charged toward your speech-to-text subscription usage.

#### **NOTE**

The long-audio segmentation service will leverage the batch transcription feature of speech-to-text, which only supports standard subscription (S0) users. During the processing of the segmentation, your audio files and the transcripts will also be sent to the Custom Speech service to refine the recognition model so the accuracy can be improved for your data. No data will be retained during this process. After the segmentation is done, only the utterances segmented and their mapping transcripts will be stored for your downloading and training.

## **Audio files**

Follow these guidelines when preparing audio for segmentation.

PROPERTY	VALUE
File format	RIFF (.wav) with a sampling rate of at least 16 khz-16-bit in PCM or .mp3 with a bit rate of at least 256 KBps, grouped into a .zip file
File name	ASCII and Unicode characters supported. No duplicate names allowed.
Audio length	Longer than 20 seconds
Archive format	.zip
Maximum archive size	2048 MB

All audio files should be grouped into a zip file. It's OK to put .wav files and .mp3 files into one audio zip. For example, you can upload a zip file containing an audio file named 'kingstory.wav', 45-second-long, and another audio named 'queenstory.mp3', 200-second-long. All .mp3 files will be transformed into the .wav format after processing.

## **Transcripts**

Transcripts must be prepared to the specifications listed in this table. Each audio file must be matched with a transcript.

PROPERTY	VALUE
File format	Plain text (.txt), grouped into a .zip
File name	Use the same name as the matching audio file
Encoding format	UTF-8-BOM only
# of utterances per line	No limit
Maximum file size	2048 MB

All transcripts files in this data type should be grouped into a zip file. For example, you have uploaded a zip file containing an audio file named 'kingstory.wav', 45 seconds long, and another one named 'queenstory.mp3', 200 seconds long. You will need to upload another zip file containing two transcripts, one named 'kingstory.txt', the other one 'queenstory.txt'. Within each plain text file, you will provide the full correct transcription for the matching audio.

After your dataset is successfully uploaded, we will help you segment the audio file into utterances based on the

transcript provided. You can check the segmented utterances and the matching transcripts by downloading the dataset. Unique IDs will be assigned to the segmented utterances automatically. It's important that you make sure the transcripts you provide are 100% accurate. Errors in the transcripts can reduce the accuracy during the audio segmentation and further introduce quality loss in the training phase that comes later.

## Audio only (beta)

If you don't have transcriptions for your audio recordings, use the **Audio only** option to upload your data. Our system can help you segment and transcribe your audio files. Keep in mind, this service will count toward your speech-to-text subscription usage.

Follow these guidelines when preparing audio.

### NOTE

The long-audio segmentation service will leverage the batch transcription feature of speech-to-text, which only supports standard subscription (S0) users.

PROPERTY	VALUE
File format	RIFF (.wav) with a sampling rate of at least 16 khz-16-bit in PCM or .mp3 with a bit rate of at least 256 Kbps, grouped into a .zip file
File name	ASCII and Unicode characters supported. No duplicate name allowed.
Audio length	Longer than 20 seconds
Archive format	.zip
Maximum archive size	2048 MB

All audio files should be grouped into a zip file. Once your dataset is successfully uploaded, we will help you segment the audio file into utterances based on our speech batch transcription service. Unique IDs will be assigned to the segmented utterances automatically. Matching transcripts will be generated through speech recognition. All .mp3 files will be transformed into the .wav format after processing. You can check the segmented utterances and the matching transcripts by downloading the dataset.

## Next steps

- [Create a Custom Voice](#)
- [Guide: Record your voice samples](#)

# Create a Custom Voice

12/4/2019 • 8 minutes to read • [Edit Online](#)

In [Prepare data for Custom Voice](#), we described the different data types you can use to train a custom voice and the different format requirements. Once you have prepared your data, you can start to upload them to the [Custom Voice portal](#), or through the Custom Voice training API. Here we describe the steps of training a custom voice through the portal.

## NOTE

This page assumes you have read [Get started with Custom Voice](#) and [Prepare data for Custom Voice](#), and have created a Custom Voice project.

Check the languages supported for custom voice: [language for customization](#).

## Upload your datasets

When you're ready to upload your data, go to the [Custom Voice portal](#). Create or select a Custom Voice project. The project must share the right language/locale and the gender properties as the data you intent to use for your voice training. For example, select `en-GB` if the audio recordings you have is done in English with a UK accent.

Go to the **Data** tab and click **Upload data**. In the wizard, select the correct data type that matches what you have prepared.

Each dataset you upload must meet the requirements for the data type that you choose. It is important to correctly format your data before it's uploaded. This ensures the data will be accurately processed by the Custom Voice service. Go to [Prepare data for Custom Voice](#) and make sure your data has been rightly formatted.

## NOTE

Free subscription (F0) users can upload two datasets simultaneously. Standard subscription (S0) users can upload five datasets simultaneously. If you reach the limit, wait until at least one of your datasets finishes importing. Then try again.

## NOTE

The maximum number of datasets allowed to be imported per subscription is 10 .zip files for free subscription (F0) users and 500 for standard subscription (S0) users.

Datasets are automatically validated once you hit the upload button. Data validation includes series of checks on the audio files to verify their file format, size, and sampling rate. Fix the errors if any and submit again. When the data-importing request is successfully initiated, you should see an entry in the data table that corresponds to the dataset you've just uploaded.

The following table shows the processing states for imported datasets:

STATE	MEANING
Processing	Your dataset has been received and is being processed.
Succeeded	Your dataset has been validated and may now be used to build a voice model.
Failed	Your dataset has been failed during processing due to many reasons, for example file errors, data problems or network issues.

After validation is complete, you can see the total number of matched utterances for each of your datasets in the **Utterances** column. If the data type you have selected requires long-audio segmentation, this column only reflects the utterances we have segmented for you either based on your transcripts or through the speech transcription service. You can further download the dataset validated to view the detail results of the utterances successfully imported and their mapping transcripts. Hint: long-audio segmentation can take more than an hour to complete data processing.

For en-US and zh-CN datasets, you can further download a report to check the pronunciation scores and the noise level for each of your recordings. The pronunciation score ranges from 0 to 100. A score below 70 normally indicates a speech error or script mismatch. A heavy accent can reduce your pronunciation score and impact the generated digital voice.

A higher signal-to-noise ratio (SNR) indicates lower noise in your audio. You can typically reach a 50+ SNR by recording at professional studios. Audio with an SNR below 20 can result in obvious noise in your generated voice.

Consider re-recording any utterances with low pronunciation scores or poor signal-to-noise ratios. If you can't re-record, you might exclude those utterances from your dataset.

## Build your custom voice model

After your dataset has been validated, you can use it to build your custom voice model.

1. Navigate to **Text-to-Speech > Custom Voice > Training**.
2. Click **Train model**.
3. Next, enter a **Name** and **Description** to help you identify this model.

Choose a name carefully. The name you enter here will be the name you use to specify the voice in your request for speech synthesis as part of the SSML input. Only letters, numbers, and a few punctuation characters such as -\_, \_ and (', ') are allowed. Use different names for different voice models.

A common use of the **Description** field is to record the names of the datasets that were used to create the model.

4. From the **Select training data** page, choose one or multiple datasets that you would like to use for training. Check the number of utterances before you submit them. You can start with any number of utterances for en-US and zh-CN voice models. For other locales, you must select more than 2,000 utterances to be able to train a voice.

### NOTE

Duplicate audio names will be removed from the training. Make sure the datasets you select do not contain the same audio names across multiple .zip files.

**TIP**

Using the datasets from the same speaker is required for quality results. When the datasets you have submitted for training contain a total number of less than 6,000 distinct utterances, you will train your voice model through the Statistical Parametric Synthesis technique. In the case where your training data exceeds a total number of 6,000 distinct utterances, you will kick off a training process with the Concatenation Synthesis technique. Normally the concatenation technology can result in more natural, and higher-fidelity voice results. [Contact the Custom Voice team](#) if you want to train a model with the latest Neural TTS technology that can produce a digital voice equivalent to the publically available [neural voices](#).

5. Click **Train** to begin creating your voice model.

The Training table displays a new entry that corresponds to this newly created model. The table also displays the status: Processing, Succeeded, Failed.

The status that's shown reflects the process of converting your dataset to a voice model, as shown here.

STATE	MEANING
Processing	Your voice model is being created.
Succeeded	Your voice model has been created and can be deployed.
Failed	Your voice model has been failed in training due to many reasons, for example unseen data problems or network issues.

Training time varies depending on the volume of audio data processed. Typical times range from about 30 minutes for hundreds of utterances to 40 hours for 20,000 utterances. Once your model training is succeeded, you can start to test it.

**NOTE**

Free subscription (F0) users can train one voice font simultaneously. Standard subscription (S0) users can train three voices simultaneously. If you reach the limit, wait until at least one of your voice fonts finishes training, and then try again.

**NOTE**

The maximum number of voice models allowed to be trained per subscription is 10 models for free subscription (F0) users and 100 for standard subscription (S0) users.

If you are using the neural voice training capability, you can select to train a model optimized for real-time streaming scenarios, or a HD neural model optimized for asynchronous [long-audio synthesis](#).

## Test your voice model

After your voice font is successfully built, you can test it before deploying it for use.

1. Navigate to **Text-to-Speech > Custom Voice > Testing**.
2. Click **Add test**.
3. Select one or multiple models that you would like to test.

4. Provide the text you want the voice(s) to speak. If you have selected to test multiple models at one time, the same text will be used for the testing for different models.

**NOTE**

The language of your text must be the same as the language of your voice font. Only successfully trained models can be tested. Only plain text is supported in this step.

5. Click **Create**.

Once you have submitted your test request, you will return to the test page. The table now includes an entry that corresponds to your new request and the status column. It can take a few minutes to synthesize speech. When the status column says **Succeeded**, you can play the audio, or download the text input (a .txt file) and audio output (a .wav file), and further audition the latter for quality.

You can also find the test results in the detail page of each models you have selected for testing. Go to the **Training** tab, and click the model name to enter the model detail page.

## Create and use a custom voice endpoint

After you've successfully created and tested your voice model, you deploy it in a custom Text-to-Speech endpoint. You then use this endpoint in place of the usual endpoint when making Text-to-Speech requests through the REST API. Your custom endpoint can be called only by the subscription that you have used to deploy the font.

To create a new custom voice endpoint, go to **Text-to-Speech > Custom Voice > Deployment**. Select **Add endpoint** and enter a **Name** and **Description** for your custom endpoint. Then select the custom voice model you would like to associate with this endpoint.

After you have clicked the **Add** button, in the endpoint table, you will see an entry for your new endpoint. It may take a few minutes to instantiate a new endpoint. When the status of the deployment is **Succeeded**, the endpoint is ready for use.

**NOTE**

Free subscription (F0) users can have only one model deployed. Standard subscription (S0) users can create up to 50 endpoints, each with its own custom voice.

**NOTE**

To use your custom voice, you must specify the voice model name, use the custom URI directly in an HTTP request, and use the same subscription to pass through the authentication of TTS service.

After your endpoint is deployed, the endpoint name appears as a link. Click the link to display information specific to your endpoint, such as the endpoint key, endpoint URL, and sample code.

Online testing of the endpoint is also available via the custom voice portal. To test your endpoint, choose **Check endpoint** from the **Endpoint detail** page. The endpoint testing page appears. Enter the text to be spoken (in either plain text or [SSML format](#) in the text box. To hear the text spoken in your custom voice font, select **Play**. This testing feature will be charged against your custom speech synthesis usage.

The custom endpoint is functionally identical to the standard endpoint that's used for text-to-speech requests. See [REST API](#) for more information.

## Next steps

- [Guide: Record your voice samples](#)
- [Text-to-Speech API reference](#)
- [Long Audio API](#)

# Record voice samples to create a custom voice

12/10/2019 • 17 minutes to read • [Edit Online](#)

Creating a high-quality production custom voice from scratch is not a casual undertaking. The central component of a custom voice is a large collection of audio samples of human speech. It's vital that these audio recordings be of high quality. Choose a voice talent who has experience making these kinds of recordings, and have them recorded by a competent recording engineer using professional equipment.

Before you can make these recordings, though, you need a script: the words that will be spoken by your voice talent to create the audio samples. For best results, your script must have good phonetic coverage and sufficient variety to train the custom voice model.

Many small but important details go into creating a professional voice recording. This guide is a roadmap for a process that will help you get good, consistent results.

## TIP

For the highest quality results, consider engaging Microsoft to help develop your custom voice. Microsoft has extensive experience producing high-quality voices for its own products, including Cortana and Office.

## Voice recording roles

There are four basic roles in a custom voice recording project:

ROLE	PURPOSE
Voice talent	This person's voice will form the basis of the custom voice.
Recording engineer	Oversees the technical aspects of the recording and operates the recording equipment.
Director	Prepares the script and coaches the voice talent's performance.
Editor	Finalizes the audio files and prepares them for upload to the Custom Voice portal.

An individual may fill more than one role. This guide assumes that you will be primarily filling the director role and hiring both a voice talent and a recording engineer. If you want to make the recordings yourself, this article includes some information about the recording engineer role. The editor role isn't needed until after the session, so can be performed by the director or the recording engineer.

## Choose your voice talent

Actors with experience in voiceover or voice character work make good custom voice talent. You can also often find suitable talent among announcers and newsreaders.

Choose voice talent whose natural voice you like. It is possible to create unique "character" voices, but it's much harder for most talent to perform them consistently, and the effort can cause voice strain.

**TIP**

Generally, avoid using recognizable voices to create a custom voice—unless, of course, your goal is to produce a celebrity voice. Lesser-known voices are usually less distracting to users.

The single most important factor for choosing voice talent is consistency. Your recordings should all sound like they were made on the same day in the same room. You can approach this ideal through good recording practices and engineering.

Your voice talent is the other half of the equation. They must be able to speak with consistent rate, volume level, pitch, and tone. Clear diction is a must. The talent also needs to be able to strictly control their pitch variation, emotional affect, and speech mannerisms.

Recording custom voice samples can be more fatiguing than other kinds of voice work. Most voice talent can record for two or three hours a day. Limit sessions to three or four a week, with a day off in-between if possible.

Recordings made for a voice model should be emotionally neutral. That is, a sad utterance should not be read in a sad way. Mood can be added to the synthesized speech later through prosody controls. Work with your voice talent to develop a "persona" that defines the overall sound and emotional tone of the custom voice. In the process, you'll pinpoint what "neutral" sounds like for that persona.

A persona might have, for example, a naturally upbeat personality. So "their" voice might carry a note of optimism even when they speak neutrally. However, such a personality trait should be subtle and consistent. Listen to readings by existing voices to get an idea of what you're aiming for.

**TIP**

Usually, you'll want to own the voice recordings you make. Your voice talent should be amenable to a work-for-hire contract for the project.

## Create a script

The starting point of any custom voice recording session is the script, which contains the utterances to be spoken by your voice talent. (The term "utterances" encompasses both full sentences and shorter phrases.)

The utterances in your script can come from anywhere: fiction, non-fiction, transcripts of speeches, news reports, and anything else available in printed form. If you want to make sure your voice does well on specific kinds of words (such as medical terminology or programming jargon), you might want to include sentences from scholarly papers or technical documents. For a brief discussion of potential legal issues, see the "[Legalities](#)" section. You can also write your own text.

Your utterances don't need to come from the same source, or the same kind of source. They don't even need to have anything to do with each other. However, if you will use set phrases (for example, "You have successfully logged in") in your speech application, make sure to include them in your script. This will give your custom voice a better chance of pronouncing those phrases well. And if you should decide to use a recording in place of synthesized speech, you'll already have it in the same voice.

While consistency is key in choosing voice talent, variety is the hallmark of a good script. Your script should include many different words and sentences with a variety of sentence lengths, structures, and moods. Every sound in the language should be represented multiple times and in numerous contexts (called *phonetic coverage*).

Furthermore, the text should incorporate all the ways that a particular sound can be represented in writing, and place each sound at varying places in the sentences. Both declarative sentences and questions should be

included and read with appropriate intonation.

It's difficult to write a script that provides *just enough* data to allow the Custom Speech portal to build a good voice. In practice, the simplest way to make a script that achieves robust phonetic coverage is to include a large number of samples. The standard voices that Microsoft provides were built from tens of thousands of utterances. You should be prepared to record a few to several thousand utterances at minimum to build a production-quality custom voice.

Check the script carefully for errors. If possible, have someone else check it too. When you run through the script with your talent, you'll probably catch a few more mistakes.

### Script format

You can write your script in Microsoft Word. The script is for use during the recording session, so you can set it up any way you find easy to work with. Create the text file that's required by the Custom Voice portal separately.

A basic script format contains three columns:

- The number of the utterance, starting at 1. Numbering makes it easy for everyone in the studio to refer to a particular utterance ("let's try number 356 again"). You can use the Word paragraph numbering feature to number the rows of the table automatically.
- A blank column where you'll write the take number or time code of each utterance to help you find it in the finished recording.
- The text of the utterance itself.

### Custom Voice Script Session Date: 6/1/65 Voice Talent: A. Lincoln

Time Code	Utterance
1	Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.
2	Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure.
3	We are met on a great battlefield of that war.
4	We have come to dedicate a portion of that field as a final resting place for those who here gave their lives that that nation might live.

#### NOTE

Most studios record in short segments known as *takes*. Each take typically contains 10 to 24 utterances. Just noting the take number is sufficient to find an utterance later. If you're recording in a studio that prefers to make longer recordings, you'll want to note the time code instead. The studio will have a prominent time display.

Leave enough space after each row to write notes. Be sure that no utterance is split between pages. Number the pages, and print your script on one side of the paper.

Print three copies of the script: one for the talent, one for the engineer, and one for the director (you). Use a paper clip instead of staples: an experienced voice artist will separate the pages to avoid making noise as the pages are turned.

### Legalities

Under copyright law, an actor's reading of copyrighted text might be a performance for which the author of the work should be compensated. This performance will not be recognizable in the final product, the custom voice. Even so, the legality of using a copyrighted work for this purpose is not well established. Microsoft cannot provide legal advice on this issue; consult your own counsel.

Fortunately, it is possible to avoid these issues entirely. There are many sources of text you can use without permission or license.

TEXT SOURCE	DESCRIPTION
CMU Arctic corpus	About 1100 sentences selected from out-of-copyright works specifically for use in speech synthesis projects. An excellent starting point.
Works no longer under copyright	Typically works published prior to 1923. For English, <a href="#">Project Gutenberg</a> offers tens of thousands of such works. You may want to focus on newer works, as the language will be closer to modern English.
Government works	Works created by the United States government are not copyrighted in the United States, though the government may claim copyright in other countries/regions.
Public domain	Works for which copyright has been explicitly disclaimed or that have been dedicated to the public domain. It may not be possible to waive copyright entirely in some jurisdictions.
Permissively-licensed works	Works distributed under a license like Creative Commons or the GNU Free Documentation License (GFDL). Wikipedia uses the GFDL. Some licenses, however, may impose restrictions on performance of the licensed content that may impact the creation of a custom voice model, so read the license carefully.

## Recording your script

Record your script at a professional recording studio that specializes in voice work. They'll have a recording booth, the right equipment, and the right people to operate it. It pays not to skimp on recording.

Discuss your project with the studio's recording engineer and listen to their advice. The recording should have little or no dynamic range compression (maximum of 4:1). It is critical that the audio have consistent volume and a high signal-to-noise ratio, while being free of unwanted sounds.

### Do it yourself

If you want to make the recording yourself, rather than going into a recording studio, here's a short primer. Thanks to the rise of home recording and podcasting, it's easier than ever to find good recording advice and resources online.

Your "recording booth" should be a small room with no noticeable echo or "room tone." It should be as quiet and soundproof as possible. Drapes on the walls can be used to reduce echo and neutralize or "deaden" the sound of the room.

Use a high-quality studio condenser microphone ("mic" for short) intended for recording voice. Sennheiser, AKG, and even newer Zoom mics can yield good results. You can buy a mic, or rent one from a local audio-visual rental firm. Look for one with a USB interface. This type of mic conveniently combines the microphone element, preamp, and analog-to-digital converter into one package, simplifying hookup.

You may also use an analog microphone. Many rental houses offer "vintage" microphones renowned for their voice character. Note that professional analog gear uses balanced XLR connectors, rather than the 1/4-inch plug that's used in consumer equipment. If you go analog, you'll also need a preamp and a computer audio interface with these connectors.

Install the microphone on a stand or boom, and install a pop filter in front of the microphone to eliminate noise from "plosive" consonants like "p" and "b." Some microphones come with a suspension mount that isolates them from vibrations in the stand, which is helpful.

The voice talent must stay at a consistent distance from the microphone. Use tape on the floor to mark where they should stand. If the talent prefers to sit, take special care to monitor mic distance and avoid chair noise.

Use a stand to hold the script. Avoid angling the stand so that it can reflect sound toward the microphone.

The person operating the recording equipment—the engineer—should be in a separate room from the talent, with some way to talk to the talent in the recording booth (a *talkback circuit*).

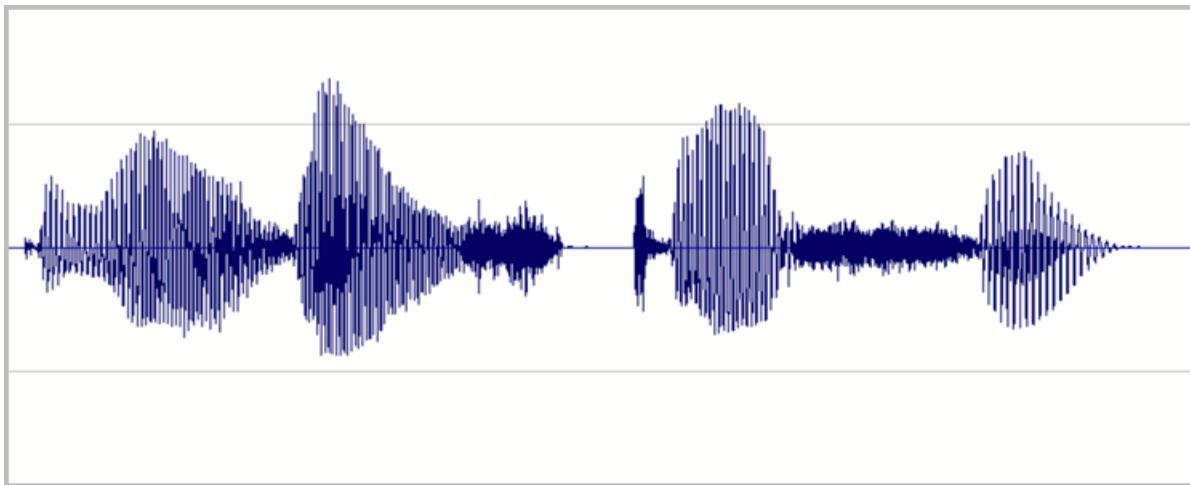
The recording should contain as little noise as possible, with a goal of an 80-db signal-to-noise ratio or better.

Listen closely to a recording of silence in your "booth," figure out where any noise is coming from, and eliminate the cause. Common sources of noise are air vents, fluorescent light ballasts, traffic on nearby roads, and equipment fans (even notebook PCs might have fans). Microphones and cables can pick up electrical noise from nearby AC wiring, usually a hum or buzz. A buzz can also be caused by a *ground loop*, which is caused by having equipment plugged into more than one electrical circuit.

**TIP**

In some cases, you might be able to use an equalizer or a noise reduction software plug-in to help remove noise from your recordings, although it is always best to stop it at its source.

Set levels so that most of the available dynamic range of digital recording is used without overdriving. That means set the audio loud, but not so loud that it becomes distorted. An example of the waveform of a good recording is shown in the following image:



Here, most of the range (height) is used, but the highest peaks of the signal do not reach the top or bottom of the window. You can also see that the silence in the recording approximates a thin horizontal line, indicating a low noise floor. This recording has acceptable dynamic range and signal-to-noise ratio.

Record directly into the computer via a high-quality audio interface or a USB port, depending on the mic you're using. For analog, keep the audio chain simple: mic, preamp, audio interface, computer. You can license both [Avid Pro Tools](#) and [Adobe Audition](#) monthly at a reasonable cost. If your budget is extremely tight, try the free [Audacity](#).

Record at 44.1 kHz 16 bit monophonic (CD quality) or better. Current state-of-the-art is 48 kHz 24-bit, if your equipment supports it. You will down-sample your audio to 16 kHz 16-bit before you submit it to the Custom Voice portal. Still, it pays to have a high-quality original recording in the event edits are needed.

Ideally, have different people serve in the roles of director, engineer, and talent. Don't try to do it all yourself. In a pinch, one person can be both the director and the engineer.

### Before the session

To avoid wasting studio time, run through the script with your voice talent before the recording session. While the voice talent becomes familiar with the text, they can clarify the pronunciation of any unfamiliar words.

#### NOTE

Most recording studios offer electronic display of scripts in the recording booth. In this case, type your run-through notes directly into the script's document. You'll still want a paper copy to take notes on during the session, though.

Most engineers will want a hard copy, too. And you'll still want a third printed copy as a backup for the talent in case the computer is down.

Your voice talent might ask which word you want emphasized in an utterance (the "operative word"). Tell them that you want a natural reading with no particular emphasis. Emphasis can be added when speech is synthesized; it should not be a part of the original recording.

Direct the talent to pronounce words distinctly. Every word of the script should be pronounced as written. Sounds should not be omitted or slurred together, as is common in casual speech, *unless they have been written that way in the script*.

WRITTEN TEXT	UNWANTED CASUAL PRONUNCIATION
never going to give you up	never gonna give you up
there are four lights	there're four lights
how's the weather today	how's th' weather today
say hello to my little friend	say hello to my lil' friend

The talent should *not* add distinct pauses between words. The sentence should still flow naturally, even while sounding a little formal. This fine distinction might take practice to get right.

### The recording session

Create a reference recording, or *match file*, of a typical utterance at the beginning of the session. Ask the talent to repeat this line every page or so. Each time, compare the new recording to the reference. This practice helps the talent remain consistent in volume, tempo, pitch, and intonation. Meanwhile, the engineer can use the match file as a reference for levels and overall consistency of sound.

The match file is especially important when you resume recording after a break or on another day. You'll want to play it a few times for the talent and have them repeat it each time until they are matching well.

Coach your talent to take a deep breath and pause for a moment before each utterance. Record a couple of seconds of silence between utterances. Words should be pronounced the same way each time they appear, considering context. For example, "record" as a verb is pronounced differently from "record" as a noun.

Record a good five seconds of silence before the first recording to capture the "room tone." This practice helps the Custom Voice portal compensate for any remaining noise in the recordings.

**TIP**

All you really need to capture is the voice talent, so you can make a monophonic (single-channel) recording of just their lines. However, if you record in stereo, you can use the second channel to record the chatter in the control room to capture discussion of particular lines or takes. Remove this track from the version that's uploaded to the Custom Voice portal.

Listen closely, using headphones, to the voice talent's performance. You're looking for good but natural diction, correct pronunciation, and a lack of unwanted sounds. Don't hesitate to ask your talent to re-record an utterance that doesn't meet these standards.

**TIP**

If you are using a large number of utterances, a single utterance might not have a noticeable effect on the resultant custom voice. It might be more expedient to simply note any utterances with issues, exclude them from your dataset, and see how your custom voice turns out. You can always go back to the studio and record the missed samples later.

Note the take number or time code on your script for each utterance. Ask the engineer to mark each utterance in the recording's metadata or cue sheet as well.

Take regular breaks and provide a beverage to help your voice talent keep their voice in good shape.

### After the session

Modern recording studios run on computers. At the end of the session, you receive one or more audio files, not a tape. These files will probably be WAV or AIFF format in CD quality (44.1 kHz 16-bit) or better. 48 kHz 24-bit is common and desirable. Higher sampling rates, such as 96 kHz, are generally not needed.

The Custom Voice portal requires each provided utterance to be in its own file. Each audio file delivered by the studio contains multiple utterances. So the primary post-production task is to split up the recordings and prepare them for submission. The recording engineer might have placed markers in the file (or provided a separate cue sheet) to indicate where each utterance starts.

Use your notes to find the exact takes you want, and then use a sound editing utility, such as [Avid Pro Tools](#), [Adobe Audition](#), or the free [Audacity](#), to copy each utterance into a new file.

Leave only about 0.2 seconds of silence at the beginning and end of each clip, except for the first. That file should start with a full five seconds of silence. Do not use an audio editor to "zero out" silent parts of the file. Including the "room tone" will help the Custom Voice algorithms compensate for any residual background noise.

Listen to each file carefully. At this stage, you can edit out small unwanted sounds that you missed during recording, like a slight lip smack before a line, but be careful not to remove any actual speech. If you can't fix a file, remove it from your dataset and note that you have done so.

Convert each file to 16 bits and a sample rate of 16 kHz before saving and, if you recorded the studio chatter, remove the second channel. Save each file in WAV format, naming the files with the utterance number from your script.

Finally, create the *transcript* that associates each WAV file with a text version of the corresponding utterance. [Creating custom voice fonts](#) includes details of the required format. You can copy the text directly from your script. Then create a Zip file of the WAV files and the text transcript.

Archive the original recordings in a safe place in case you need them later. Preserve your script and notes, too.

## Next steps

You're ready to upload your recordings and create your custom voice.

[Create custom voice fonts](#)

# Audio Content Creation

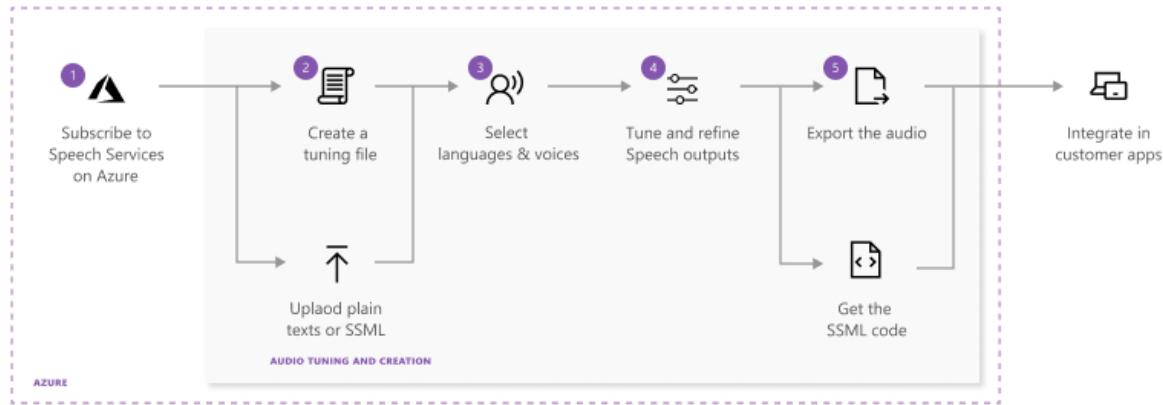
12/4/2019 • 3 minutes to read • [Edit Online](#)

[Audio Content Creation](#) is an online tool that allows you to customize and fine-tune Microsoft's text-to-speech output for your apps and products. You can use this tool to fine-tune public and custom voices for more accurate natural expressions, and manage your output in the cloud.

The Audio Content Creation tool is based on [Speech Synthesis Markup Language \(SSML\)](#). To simplify customization and tuning, Audio Content Creation allows you to visually inspect your text-to-speech outputs in real time.

## How does it work?

This diagram shows the steps it takes to tune and export customized speech-to-text outputs. Use the links below to learn more about each step.



1. The first step is to [create an Azure account, register a Speech resource, and get a subscription key](#). After you have a subscription key, you can use it to call the Speech service, and to access [Audio Content Creation](#).
2. [Create an audio tuning file](#) using plain text or SSML.
3. Choose the voice and the language that you'd like to tune. Audio Content Creation includes all of the [Microsoft text-to-speech voices](#). You can use standard, neural, or your own custom voice.

### NOTE

Gated access is available for Custom Neural Voices, which allow you to create high-definition voices similar to natural-sounding speech. For additional details, see [Gating process](#).

4. Review the default result. Then use the tuning tool to adjust pronunciation, pitch, rate, intonation, voice style, and more. For a complete list of options, see [Speech Synthesis Markup Language](#).
5. Save and [export your tuned audio](#). When you save the tuning track in the system, you can continue to work and iterate on the output. When you're satisfied with the output, you can create an audio creation task with the export feature. You can observe the status of the export task, and download the output for use with your apps and products.

6. The last step is to use the custom tuned voice in your apps and products.

## Create a Speech resource

Follow these steps to create a Speech resource and connect it with Speech Studio.

1. Follow these instructions to [sign up for an Azure account](#) and [create a Speech resource](#). Make sure that your pricing tier is set to **S0**. If you are using one of the Neural voices, make sure that you create your resource in a [supported region](#).
2. Sign into [Audio Content Creation](#).
3. Select an existing project, or click **Create New**.
4. You can modify your subscription at any time with the **Settings** option, located in the top nav.

## Create an audio tuning file

There are two ways to get your content into the Audio Content Creation tool.

### Option 1:

1. After you sign into [Audio Content Creation](#), click **Audio Tuning** to create a new audio tuning file.
2. When the editing window appears, you can input up to 10,000 characters.
3. Don't forget to save.

### Option 2:

1. After you sign into [Audio Content Creation](#), click **Upload** to import one or more text files. Both plain text and SSML are supported.
2. When you upload your text files, make sure that the content meets these requirements.

PROPERTY	VALUE / NOTES
File format	Plain text (.txt) SSML text (.txt) Zip files aren't supported
Encoding format	UTF-8
File name	Each file must have a unique name. Duplicates aren't supported.
Text length	Text files must not exceed 10,000 characters.
SSML restrictions	Each SSML file can only contain a single piece of SSML.

### Plain text example

Welcome to use Audio Content Creation to customize audio output for your products.

### SSML text example

```
<speak xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" version="1.0"
xml:lang="en-US">
 <voice name="Microsoft Server Speech Text to Speech Voice (en-US, JessaNeural)">
 Welcome to use Audio Content Creation <break time="10ms" />to customize audio output for your products.
 </voice>
</speak>
```

## Export tuned audio

After you've reviewed your audio output and are satisfied with your tuning and adjustment, you can export the audio.

1. From the [Audio Content Creation](#) tool, click **Export** to create an audio creation task.
2. Choose the output format for your tuned audio. A list of supported formats and sample rates is available below.
3. You can view the status of the task on the **Export task** tab. If the task fails, see the detailed information page for a full report.
4. When the task is complete, your audio is available for download on the **Audio Library** tab.
5. Click **Download**. Now you're ready to use your custom tuned audio in your apps or products.

### Supported audio formats

FORMAT	16 KHZ SAMPLE RATE	24 KHZ SAMPLE RATE
wav	riff-16khz-16bit-mono-pcm	riff-24khz-16bit-mono-pcm
mp3	audio-16khz-128kbitrate-mono-mp3	audio-24khz-160kbitrate-mono-mp3

## See also

- [Long Audio API](#)

## Next steps

[Speech Studio](#)

# Guidelines for creating an effective keyword

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

Creating an effective keyword is vital to ensuring your device will consistently and accurately respond. Customizing your keyword is an effective way to differentiate your device and strengthen your branding. In this article, you learn some guiding principles for creating an effective keyword.

## Choose an effective keyword

Consider the following guidelines when you choose a keyword:

- Your keyword should be an English word or phrase.
- It should take no longer than two seconds to say.
- Words of 4 to 7 syllables work best. For example, "Hey, Computer" is a good keyword. Just "Hey" is a poor one.
- Keywords should follow common English pronunciation rules.
- A unique or even a made-up word that follows common English pronunciation rules might reduce false positives. For example, "computerama" might be a good keyword.
- Do not choose a common word. For example, "eat" and "go" are words that people say frequently in ordinary conversation. They might be false triggers for your device.
- Avoid using a keyword that might have alternative pronunciations. Users would have to know the "right" pronunciation to get their device to respond. For example, "509" can be pronounced "five zero nine," "five oh nine," or "five hundred and nine." "R.E.I." can be pronounced "r-e-i" or "ray." "Live" can be pronounced "/liv/" or "/liv/'.
- Do not use special characters, symbols, or digits. For example, "Go#" and "20 + cats" could be problematic keywords. However, "go sharp" or "twenty plus cats" might work. You can still use the symbols in your branding and use marketing and documentation to reinforce the proper pronunciation.

### NOTE

If you choose a trademarked word as your keyword, be sure that you own that trademark or that you have permission from the trademark owner to use the word. Microsoft is not liable for any legal issues that might arise from your choice of keyword.

## Next steps

Learn how to [create a custom keyword using Speech Studio](#).

# Create a custom keyword using Speech Studio

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

Your device is always listening for a keyword (or phrase). For example, "Hey Cortana" is a keyword for the Cortana assistant. When the user says the keyword, the device sends all subsequent audio to the cloud, until the user stops speaking. Customizing your keyword is an effective way to differentiate your device and strengthen your branding.

In this article, you learn how to create a custom keyword for your device.

## Create your keyword

Before you can use a custom keyword, you'll need to create a keyword using the [Custom Keyword](#) page on [Speech Studio](#). After you provide a keyword, it produces a file that you deploy to your device.

1. Go to the [Speech Studio](#) and [Sign in](#) or, if you do not yet have a speech subscription, choose [Create a subscription](#).
2. At the [Custom Keyword](#) page, create a [New project](#).
3. Enter a **Name**, an optional **Description**, and select the language. You will need one project per language and support is currently limited to the en-US language.

The screenshot shows a 'New project' dialog box with the following fields:

- Name \***: A text input field containing 'Name your project'.
- Description**: A text input field containing 'Describe your project'.
- Language \***: A dropdown menu with 'Select a language' placeholder text.

Note: Your language should match the data you intend to use for training.

Buttons at the bottom right: **Cancel** (blue outline) and **Create** (grey).

4. Select your project from the list.

## Speech Studio

### Custom Keyword

Create your own custom keyword to work with our [Speech Device SDK](#). Please read the guide to select an effective keyword. [Learn more](#)

[New project](#)  

Name	Description	Language	Datasets	Models	Created
<input type="checkbox"/> MyKeyword	A test keyword	English (United States)	0	0	11/27/2019 9:16 AM

5. To start a new keyword model click **Train model**.
6. Enter a **Name** for the keyword model, and optional **Description** and type in the **Keyword** of your choice, and click **Next**. We have some [guidelines](#) to help choose an effective keyword.

New model X

Name \*

Description

Keyword \*

[Cancel](#) [Next](#)

7. The portal will now create candidate pronunciations for your keyword. Listen to each candidate by clicking the play buttons and remove the checks next to any pronunciations that are incorrect. Once only good pronunciations are checked, click **Train** to begin generating the keyword.

Choose pronunciations X

Keyword

Hello Computer

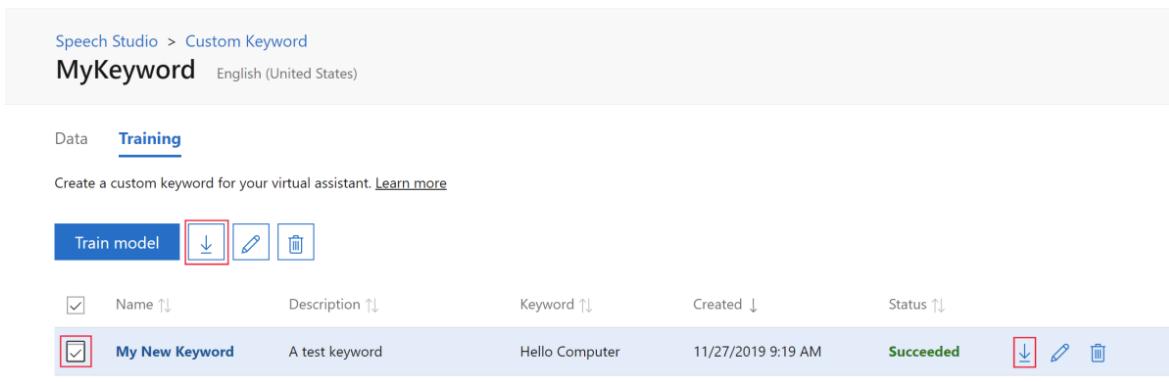
Select pronunciations

Add training data to train keyword model.

[Back](#) [Train](#)

8. It may take up to ten minutes for the model to be generated. The keyword list will change from **Processing** to **Succeeded** when the model is complete. You can then download the file.



The screenshot shows the 'Custom Keyword' section of the Speech Studio interface. At the top, it says 'Speech Studio > Custom Keyword' and 'MyKeyword English (United States)'. Below this, there are two tabs: 'Data' and 'Training', with 'Training' being the active tab. A sub-instruction 'Create a custom keyword for your virtual assistant.' is followed by a link 'Learn more'. There are three buttons: 'Train model' (blue), a download icon (red box), a edit icon, and a delete icon. A table lists one keyword: 'My New Keyword' (checkbox checked, red box), 'A test keyword' (Description), 'Hello Computer' (Keyword), '11/27/2019 9:19 AM' (Created), and 'Succeeded' (Status). To the right of the table are three icons: download, edit, and delete.

Name ↑	Description ↑↑	Keyword ↑↓	Created ↓	Status ↑↓
<input checked="" type="checkbox"/> My New Keyword	A test keyword	Hello Computer	11/27/2019 9:19 AM	Succeeded

9. Save the .zip file to your computer. You will need this file to deploy your custom keyword to your device.

## Next steps

Test your custom keyword with [Speech Devices SDK Quickstart](#).

# Speech service for telephony data

12/4/2019 • 10 minutes to read • [Edit Online](#)

Telephony data that is generated through landlines, mobile phones, and radios are typically low quality, and narrowband in the range of 8 KHz, which creates challenges when converting speech-to-text. The latest speech recognition models from the Speech service excel at transcribing this telephony data, even in cases when the data is difficult for a human to understand. These models are trained with large volumes of telephony data, and have best-in-market recognition accuracy, even in noisy environments.

A common scenario for speech-to-text is transcribing large volumes of telephony data that may come from various systems, such as Interactive Voice Response (IVR). The audio these systems provide can be stereo or mono, and raw with little-to-no post processing done on the signal. Using the Speech service and the Unified speech model, a business can get high-quality transcriptions, whatever systems are used to capture audio.

Telephony data can be used to better understand your customers' needs, identify new marketing opportunities, or evaluate the performance of call center agents. After the data is transcribed, a business can use the output for purposes such as improved telemetry, identifying key phrases, or analyzing customer sentiment.

The technologies outlined in this page are by Microsoft internally for various support call processing services, both in real-time and batch mode.

Let's review some of the technology and related features the Speech service offers.

## IMPORTANT

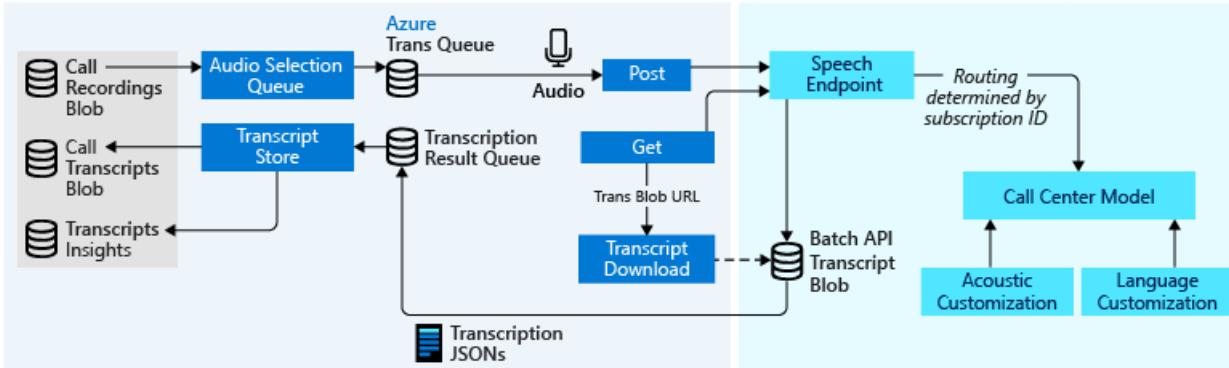
The Speech service Unified model is trained with diverse data and offers a single-model solution to a number of scenarios from Dictation to Telephony analytics.

## Azure Technology for Call Centers

Beyond the functional aspect of the Speech service features, their primary purpose – when applied to the call center – is to improve the customer experience. Three clear domains exist in this regard:

- Post-call analytics, which is essentially batch processing of call recordings after the call.
- Real-time analytics, which is processing of the audio signal to extract various insights as the call is taking place (with sentiment being a prominent use case).
- Voice assistants (bots), either driving the dialogue between the customer and the bot in an attempt to solve the customer's issue with no agent participation, or being the application of artificial intelligence (AI) protocols to assist the agent.

A typical architecture diagram of the implementation of a batch scenario is depicted in the picture below



## Speech Analytics Technology Components

Whether the domain is post-call or real-time, Azure offers a set of mature and emerging technologies to improve the customer experience.

### Speech to text (STT)

[Speech-to-text](#) is the most sought-after feature in any call center solution. Because many of the downstream analytics processes rely on transcribed text, the word error rate (*WER*) is of utmost importance. One of the key challenges in call center transcription is the noise that's prevalent in the call center (for example other agents speaking in the background), the rich variety of language locales and dialects as well as the low quality of the actual telephone signal. *WER* is highly correlated with how well the acoustic and language models are trained for a given locale, thus the ability to customize the model to your locale is important. Our latest Unified version 4.x models are the solution to both transcription accuracy and latency. Trained with tens of thousands of hours of acoustic data and billions of lexical information, Unified models are the most accurate models in the market to transcribe call center data.

### Sentiment

Gauging whether the customer had a good experience is one of the most important areas of Speech analytics when applied to the call center space. Our [Batch Transcription API](#) offers sentiment analysis per utterance. You can aggregate the set of values obtained as part of a call transcript to determine the sentiment of the call for both your agents and the customer.

### Silence (non-talk)

It is not uncommon for 35 percent of a support call to be what we call non-talk time. Some scenarios for which non-talk occurs are: agents looking up prior case history with a customer, agents using tools that allow them to access the customer's desktop and perform functions, customers sitting on hold waiting for a transfer, and so on. It is extremely important to gauge when silence is occurring in a call as there are number of important customer sensitivities that occur around these types of scenarios and where they occur in the call.

### Translation

Some companies are experimenting with providing translated transcripts from foreign language support calls so that delivery managers can understand the world-wide experience of their customers. Our [translation](#) capabilities are unsurpassed. We can translate audio-to-audio or audio-to-text for a large number of locales.

### Text to Speech

[Text-to-speech](#) is another important area in implementing bots that interact with the customers. The typical pathway is that the customer speaks, their voice is transcribed to text, the text is analyzed for intents, a response is synthesized based on the recognized intent, and then an asset is either surfaced to the customer or a synthesized voice response is generated. Of course all of this has to occur quickly – thus low-latency is an important component in the success of these systems.

Our end-to-end latency is considerably low for the various technologies involved such as [Speech-to-text](#), [LUIS](#), [Bot](#)

## Framework, Text-to-speech.

Our new voices are also indistinguishable from human voices. You can use our voices to give your bot its unique personality.

## Search

Another staple of analytics is to identify interactions where a specific event or experience has occurred. This is typically done with one of two approaches; either an ad hoc search where the user simply types a phrase and the system responds, or a more structured query where an analyst can create a set of logical statements that identify a scenario in a call, and then each call can be indexed against that set of queries. A good search example is the ubiquitous compliance statement "this call shall be recorded for quality purposes... ". Many companies want to make sure that their agents are providing this disclaimer to customers before the call is actually recorded. Most analytics systems have the ability to trend the behaviors found by query/search algorithms, and this reporting of trends is ultimately one of the most important functions of an analytics system. Through [Cognitive services directory](#) your end to end solution can be significantly enhanced with indexing and search capabilities.

## Key Phrase Extraction

This area is one of the more challenging analytics applications and one that is benefiting from the application of AI and machine learning. The primary scenario in this case is to infer customer intent. Why is the customer calling? What is the customer problem? Why did the customer have a negative experience? Our [text analytics service](#) provides a set of analytics out of the box for quickly upgrading your end-to-end solution for extracting those important keywords or phrases.

Let's now have a look at the batch processing and the real-time pipelines for speech recognition in a bit more detail.

## Batch transcription of call center data

For transcribing bulk audio we developed the [Batch Transcription API](#). The Batch Transcription API was developed to transcribe large amounts of audio data asynchronously. With regard to transcribing call center data, our solution is based on these pillars:

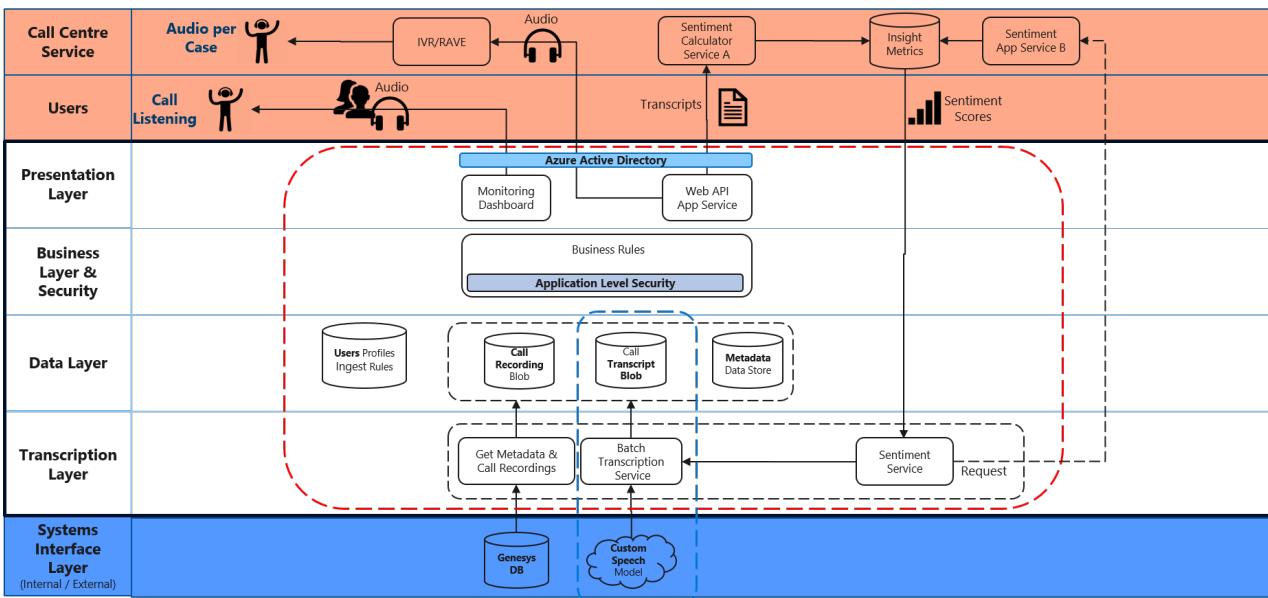
- **Accuracy** - With fourth-generation Unified models, we offer unsurpassed transcription quality.
- **Latency** - We understand that when doing bulk transcriptions, the transcriptions are needed quickly. The transcription jobs initiated via the [Batch Transcription API](#) will be queued immediately, and once the job starts running it's performed faster than real-time transcription.
- **Security** - We understand that calls may contain sensitive data. Rest assured that security is one of our highest priorities. Our service has obtained ISO, SOC, HIPAA, PCI certifications.

Call centers generate large volumes of audio data on a daily basis. If your business stores telephony data in a central location, such as Azure Storage, you can use the [Batch Transcription API](#) to asynchronously request and receive transcriptions.

A typical solution uses these services:

- The Speech service is used to transcribe speech-to-text. A standard subscription (S0) for the Speech service is required to use the Batch Transcription API. Free subscriptions (F0) will not work.
- [Azure Storage](#) is used to store telephony data, and the transcripts returned by the Batch Transcription API. This storage account should use notifications, specifically for when new files are added. These notifications are used to trigger the transcription process.
- [Azure Functions](#) is used to create the shared access signatures (SAS) URI for each recording, and trigger the HTTP POST request to start a transcription. Additionally, Azure Functions is used to create requests to retrieve and delete transcriptions using the Batch Transcription API.

Internally we are using the above technologies to support Microsoft customer calls in Batch mode.

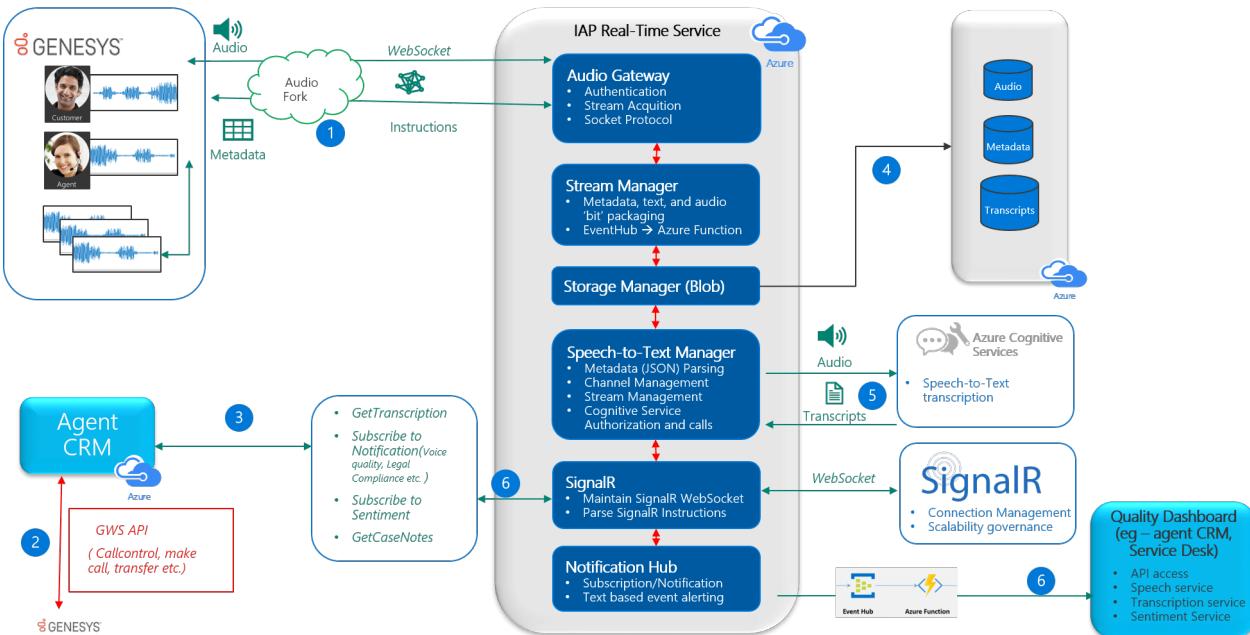


## Real-time transcription for call center data

Some businesses are required to transcribe conversations in real-time. Real-time transcription can be used to identify key-words and trigger searches for content and resources relevant to the conversation, for monitoring sentiment, to improve accessibility, or to provide translations for customers and agents who aren't native speakers.

For scenarios that require real-time transcription, we recommend using the [Speech SDK](#). Currently, speech-to-text is available in [more than 20 languages](#), and the SDK is available in C++, C#, Java, Python, Node.js, Objective-C, and JavaScript. Samples are available in each language on [GitHub](#). For the latest news and updates, see [Release notes](#).

Internally we are using the above technologies to analyze in real-time Microsoft customer calls as they happen, as illustrated in the following diagram.



## A word on IVRs

The Speech service can be easily integrated in any solution by using either the [Speech SDK](#) or the [REST API](#). However, call center transcription may require additional technologies. Typically, a connection between an IVR system and Azure is required. Although we do not offer such components, here is a description what a connection

to an IVR entails.

Several IVR or telephony service products (such as Genesys or AudioCodes) offer integration capabilities that can be leveraged to enable inbound and outbound audio pass-through to an Azure service. Basically, a custom Azure service might provide a specific interface to define phone call sessions (such as Call Start or Call End) and expose a WebSocket API to receive inbound stream audio that is used with the Speech service. Outbound responses, such as conversation transcription or connections with the Bot Framework, can be synthesized with Microsoft's text-to-speech service and returned to the IVR for playback.

Another scenario is direct integration with Session Initiation Protocol (SIP). An Azure service connects to a SIP Server, thus getting an inbound stream and an outbound stream, which is used for the speech-to-text and text-to-speech phases. To connect to a SIP Server there are commercial software offerings, such as Ozeki SDK, or [the Teams calling and meetings API](#) (currently in beta), that are designed to support this type of scenario for audio calls.

## Customize existing experiences

The Speech service works well with built-in models. However, you may want to further customize and tune the experience for your product or environment. Customization options range from acoustic model tuning to unique voice fonts for your brand. After you've built a custom model, you can use it with any of the Speech service features in real-time or batch mode.

SPEECH SERVICE	MODEL	DESCRIPTION
Speech-to-text	<a href="#">Acoustic model</a>	Create a custom acoustic model for applications, tools, or devices that are used in particular environments like in a car or on a factory floor, each with specific recording conditions. Examples include accented speech, specific background noises, or using a specific microphone for recording.
	<a href="#">Language model</a>	Create a custom language model to improve transcription of industry-specific vocabulary and grammar, such as medical terminology, or IT jargon.
	<a href="#">Pronunciation model</a>	With a custom pronunciation model, you can define the phonetic form and display for a word or term. It's useful for handling customized terms, such as product names or acronyms. All you need to get started is a pronunciation file, which is a simple <code>.txt</code> file.
Text-to-speech	<a href="#">Voice font</a>	Custom voice fonts allow you to create a recognizable, one-of-a-kind voice for your brand. It only takes a small amount of data to get started. The more data that you provide, the more natural and human-like your voice font will sound.

## Sample code

Sample code is available on GitHub for each of the Speech service features. These samples cover common

scenarios like reading audio from a file or stream, continuous and single-shot recognition, and working with custom models. Use these links to view SDK and REST samples:

- [Speech-to-text and speech translation samples \(SDK\)](#)
- [Batch transcription samples \(REST\)](#)
- [Text-to-speech samples \(REST\)](#)

## Reference docs

- [Speech SDK](#)
- [Speech Devices SDK](#)
- [REST API: Speech-to-text](#)
- [REST API: Text-to-speech](#)
- [REST API: Batch transcription and customization](#)

## Next steps

[Get a Speech service subscription key for free](#)

# Try the Speech service for free

12/4/2019 • 4 minutes to read • [Edit Online](#)

Using the Speech service is easy and affordable. There are two options available free of charge so you can discover what the service can do and decide whether it's right for your needs:

- Get a free trial without providing any credit card info (you need to have an existing Azure account)
- Create a new Azure account at no charge for a trial period (credit card information required)

In this article, you'll choose one of these options that suits your needs best.

## NOTE

The Speech service has two service tiers: free and subscription, which have different limitations and benefits. When you sign up for a free Azure account it comes with \$200 in service credit that you can apply toward a paid Speech service subscription, valid for up to 30 days.

If you use the free, low-volume Speech service tier you can keep this free subscription even after your free trial or service credit expires.

For more information, see [Cognitive Services pricing - Speech service](#).

## Try the Speech service without credit card info

Although we recommend trying the Speech service using the instructions in the next section, you may prefer this section's instructions if the following applies:

- You already have an active Azure account.
- You want to evaluate the Speech service without creating a new Azure account.
- You prefer no credit card be required and no data saved after trial period.

## NOTE

Your trial period starts immediately once the following steps are complete.

1. Go to [Try Cognitive Service](#).
2. Select the **Speech APIs** tab.
3. Choose **Get API Key**.

You'll be presented with billing choices. Choose the free option and then read and approve the user agreement. You'll be presented with keys you can use to try the Speech service for a limited time.

## Try the Speech service using a new Azure account

To sign up for a new Azure account, you will need a Microsoft account. If you do not have a Microsoft account, you can sign up for one free of charge at the [Microsoft account portal](#). Select **Sign in with Microsoft** and then, when asked to sign in, select **Create a Microsoft account**. Follow the steps to create and verify your new Microsoft account.

Once you have a Microsoft account, go to the [Azure sign-up page](#), select **Start free**, and create a new Azure account using a Microsoft account.

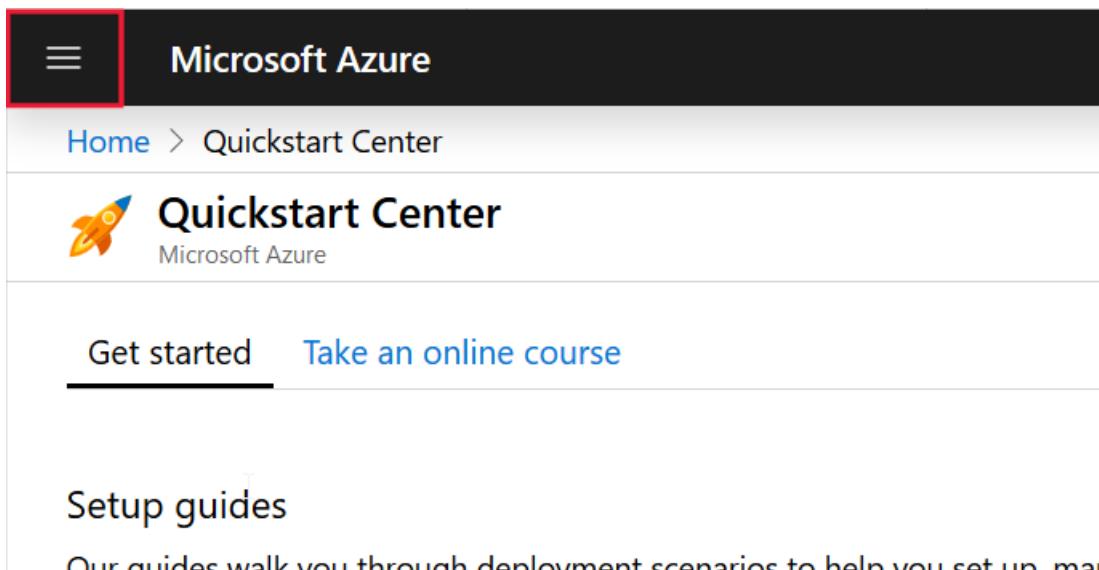
## Create a Speech resource in Azure

### NOTE

You can create an unlimited number of standard-tier subscriptions in one or multiple regions. However, you can create only one free-tier subscription. Model deployments on the free tier that remain unused for 7 days will be decommissioned automatically.

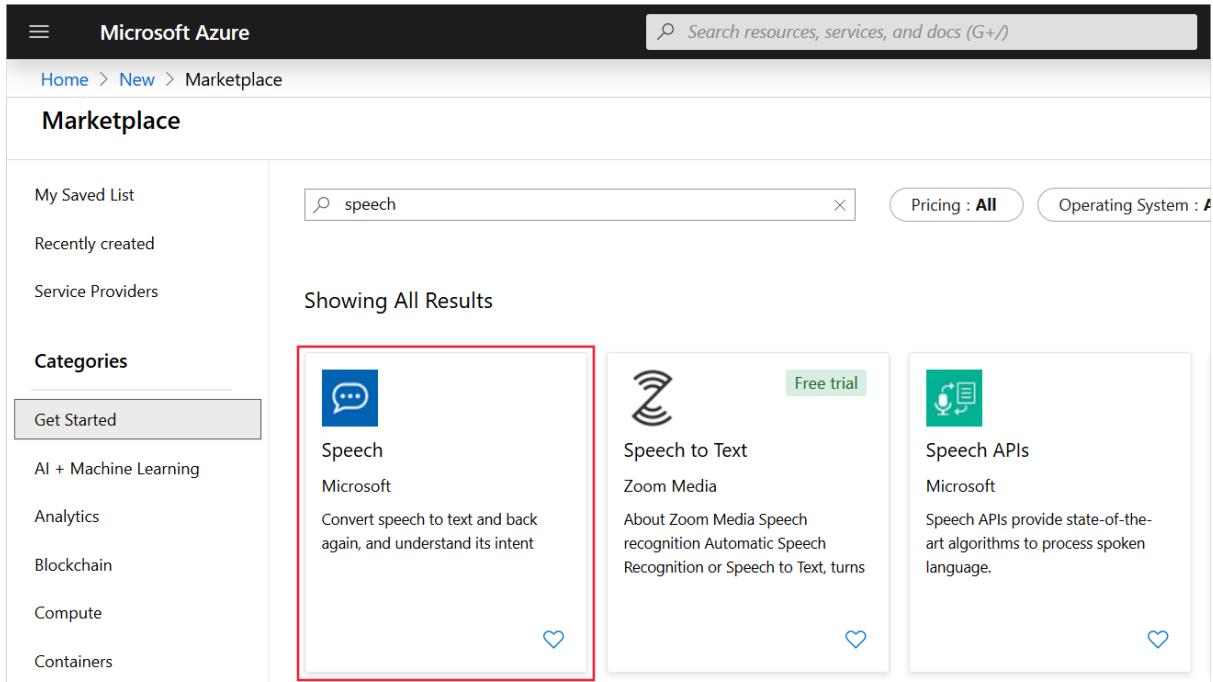
To add a Speech service resource (free or paid tier) to your Azure account:

1. Sign in to the [Azure portal](#) using your Microsoft account.
2. Select **Create a resource** at the top left of the portal. If you do not see **Create a resource**, you can always find it by selecting the collapsed menu in the upper left:



The screenshot shows the Microsoft Azure Quickstart Center. At the top, there's a navigation bar with a three-line menu icon, the text "Microsoft Azure", and a "Home" link followed by a "Quickstart Center" link. Below this is a section titled "Quickstart Center" with a rocket ship icon and "Microsoft Azure" text. There are two buttons: "Get started" (underlined) and "Take an online course". Further down, there's a section titled "Setup guides" with a sub-section about guides for development scenarios. A red box highlights the "Create a resource" button in the top-left corner of the main content area.

3. In the **New** window, type "speech" in the search box and press ENTER.
4. In the search results, select **Speech**.



The screenshot shows the Microsoft Azure Marketplace search results for "speech". The search bar at the top contains "speech". On the left, there's a sidebar with "My Saved List", "Recently created", "Service Providers", and a "Categories" section with "Get Started" selected. The main area shows "Showing All Results" with three cards: "Speech" (selected), "Speech to Text", and "Speech APIs". The "Speech" card has a "Free trial" badge and a description: "Convert speech to text and back again, and understand its intent". A red box highlights the "Speech" card.

5. Select **Create**, then:

- Give a unique name for your new resource. The name helps you distinguish among multiple

subscriptions to the same service.

- Choose the Azure subscription that the new resource is associated with to determine how the fees are billed.
- Choose the [region](#) where the resource will be used.
- Choose either a free (F0) or paid (S0) pricing tier. For complete information about pricing and usage quotas for each tier, select **View full pricing details**.
- Create a new resource group for this Speech subscription or assign the subscription to an existing resource group. Resource groups help you keep your various Azure subscriptions organized.
- Select **Create**. This will take you to the deployment overview and display deployment progress messages.

It takes a few moments to deploy your new Speech resource. Once deployment is complete, select **Go to resource** and in the left navigation pane select **Keys** to display your Speech service subscription keys. Each subscription has two keys; you can use either key in your application. To quickly copy/paste a key to your code editor or other location, select the copy button next to each key, switch windows to paste the clipboard contents to the desired location.

#### IMPORTANT

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely— for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

## Switch to a new subscription

To switch from one subscription to another, for example when your free trial expires or when you publish your application, replace the region and subscription key in your code with the region and subscription key of the new Azure resource.

## About regions

- If your application uses a [Speech SDK](#), you provide the region code, such as `westus`, when creating a speech configuration.
- If your application uses one of the Speech service's [REST APIs](#), the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

## Next steps

Complete one of our 10-minute quickstarts or check out our SDK samples:

[Quickstart: Recognize speech in C# Speech SDK samples](#)

# Try the Speech service for free

12/4/2019 • 4 minutes to read • [Edit Online](#)

Using the Speech service is easy and affordable. There are two options available free of charge so you can discover what the service can do and decide whether it's right for your needs:

- Get a free trial without providing any credit card info (you need to have an existing Azure account)
- Create a new Azure account at no charge for a trial period (credit card information required)

In this article, you'll choose one of these options that suits your needs best.

## NOTE

The Speech service has two service tiers: free and subscription, which have different limitations and benefits. When you sign up for a free Azure account it comes with \$200 in service credit that you can apply toward a paid Speech service subscription, valid for up to 30 days.

If you use the free, low-volume Speech service tier you can keep this free subscription even after your free trial or service credit expires.

For more information, see [Cognitive Services pricing - Speech service](#).

## Try the Speech service without credit card info

Although we recommend trying the Speech service using the instructions in the next section, you may prefer this section's instructions if the following applies:

- You already have an active Azure account.
- You want to evaluate the Speech service without creating a new Azure account.
- You prefer no credit card be required and no data saved after trial period.

## NOTE

Your trial period starts immediately once the following steps are complete.

1. Go to [Try Cognitive Service](#).
2. Select the **Speech APIs** tab.
3. Choose **Get API Key**.

You'll be presented with billing choices. Choose the free option and then read and approve the user agreement. You'll be presented with keys you can use to try the Speech service for a limited time.

## Try the Speech service using a new Azure account

To sign up for a new Azure account, you will need a Microsoft account. If you do not have a Microsoft account, you can sign up for one free of charge at the [Microsoft account portal](#). Select **Sign in with Microsoft** and then, when asked to sign in, select **Create a Microsoft account**. Follow the steps to create and verify your new Microsoft account.

Once you have a Microsoft account, go to the [Azure sign-up page](#), select **Start free**, and create a new Azure account using a Microsoft account.

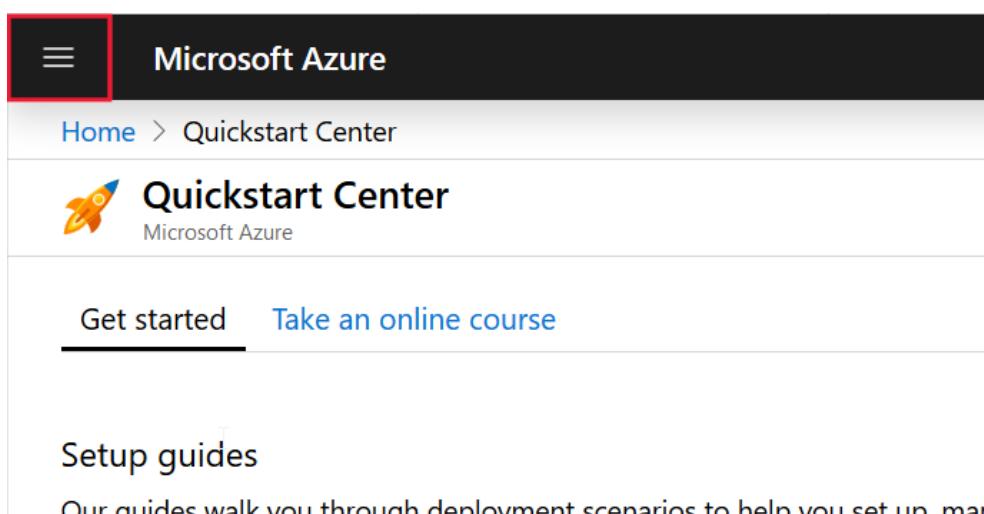
## Create a Speech resource in Azure

### NOTE

You can create an unlimited number of standard-tier subscriptions in one or multiple regions. However, you can create only one free-tier subscription. Model deployments on the free tier that remain unused for 7 days will be decommissioned automatically.

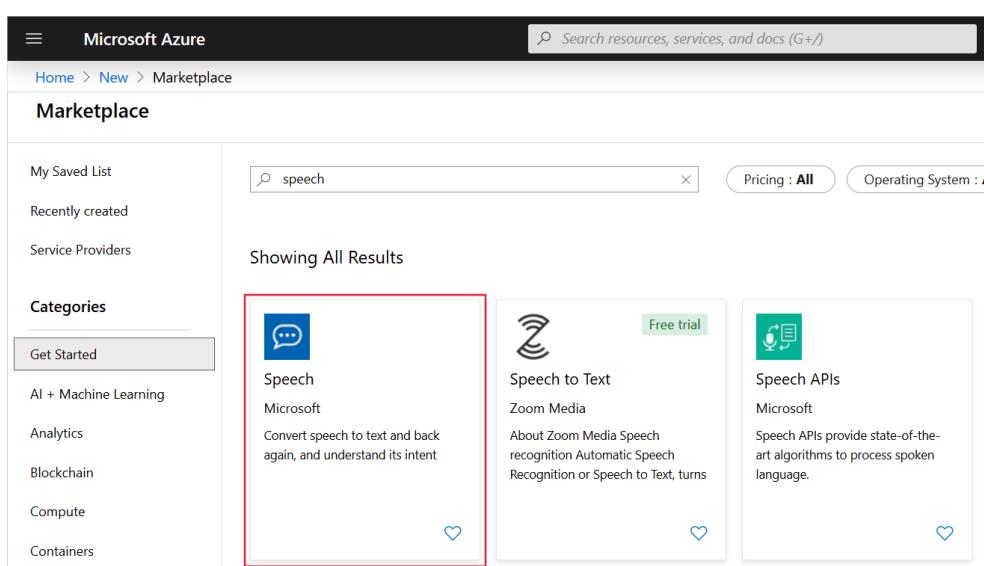
To add a Speech service resource (free or paid tier) to your Azure account:

1. Sign in to the [Azure portal](#) using your Microsoft account.
2. Select **Create a resource** at the top left of the portal. If you do not see **Create a resource**, you can always find it by selecting the collapsed menu in the upper left:



The screenshot shows the Microsoft Azure Quickstart Center. At the top, there's a navigation bar with a menu icon and the text "Microsoft Azure". Below it, a breadcrumb trail says "Home > Quickstart Center". A "Quickstart Center" section features a rocket ship icon and the text "Microsoft Azure". Below this, there are two buttons: "Get started" (which is highlighted with a red box) and "Take an online course". Further down, there's a "Setup guides" section with the text "Our guides walk you through deployment scenarios to help you set up mar...".

3. In the **New** window, type "speech" in the search box and press ENTER.
4. In the search results, select **Speech**.



The screenshot shows the Microsoft Azure Marketplace search results for "speech". The search bar at the top contains "speech". On the left, there's a sidebar with "My Saved List", "Recently created", "Service Providers", and a "Categories" section with "Get Started" selected, along with other options like "AI + Machine Learning", "Analytics", "Blockchain", "Compute", and "Containers". The main area shows a grid of service cards. One card for "Speech" from Microsoft is highlighted with a red box. Other visible cards include "Speech to Text" (with a "Free trial" badge), "Zoom Media", and "Speech APIs". Each card has a brief description and a "Free trial" or "Pricing: All" badge.

5. Select **Create**, then:
  - Give a unique name for your new resource. The name helps you distinguish among multiple subscriptions to the same service.
  - Choose the Azure subscription that the new resource is associated with to determine

how the fees are billed.

- Choose the [region](#) where the resource will be used.
- Choose either a free (F0) or paid (S0) pricing tier. For complete information about pricing and usage quotas for each tier, select **View full pricing details**.
- Create a new resource group for this Speech subscription or assign the subscription to an existing resource group. Resource groups help you keep your various Azure subscriptions organized.
- Select **Create**. This will take you to the deployment overview and display deployment progress messages.

It takes a few moments to deploy your new Speech resource. Once deployment is complete, select **Go to resource** and in the left navigation pane select **Keys** to display your Speech service subscription keys. Each subscription has two keys; you can use either key in your application. To quickly copy/paste a key to your code editor or other location, select the copy button next to each key, switch windows to paste the clipboard contents to the desired location.

#### IMPORTANT

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

## Switch to a new subscription

To switch from one subscription to another, for example when your free trial expires or when you publish your application, replace the region and subscription key in your code with the region and subscription key of the new Azure resource.

## About regions

- If your application uses a [Speech SDK](#), you provide the region code, such as `westus`, when creating a speech configuration.
- If your application uses one of the Speech service's [REST APIs](#), the region is part of the endpoint URI you use when making requests.
- Keys created for a region are valid only in that region. Attempting to use them with other regions will result in authentication errors.

## Next steps

Complete one of our 10-minute quickstarts or check out our SDK samples:

[Quickstart: Recognize speech in C# Speech SDK samples](#)

# Install and run Speech service containers (Preview)

12/4/2019 • 17 minutes to read • [Edit Online](#)

Containers enable you to run some of the Speech service APIs in your own environment. Containers are great for specific security and data governance requirements. In this article you'll learn how to download, install, and run a Speech container.

Speech containers enable customers to build a speech application architecture that is optimized for both robust cloud capabilities and edge locality. There are four different containers available. The two standard containers are **Speech-to-text** and **Text-to-speech**. The two custom containers are **Custom Speech-to-text** and **Custom Text-to-speech**.

## IMPORTANT

All speech containers are currently offered as part of a [Public "Gated" Preview](#). An announcement will be made when speech containers progress to General Availability (GA).

FUNCTION	FEATURES	LATEST
Speech-to-text	Transcribes continuous real-time speech or batch audio recordings into text with intermediate results.	2.0.0
Custom Speech-to-text	Using a custom model from the <a href="#">Custom Speech portal</a> , transcribes continuous real-time speech or batch audio recordings into text with intermediate results.	2.0.0
Text-to-speech	Converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.3.0
Custom Text-to-speech	Using a custom model from the <a href="#">Custom Voice portal</a> , converts text to natural-sounding speech with plain text input or Speech Synthesis Markup Language (SSML).	1.3.0

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Prerequisites

The following prerequisites before using Speech containers:

REQUIRED	PURPOSE
----------	---------

REQUIRED	PURPOSE
Docker Engine	<p>You need the Docker Engine installed on a <a href="#">host computer</a>. Docker provides packages that configure the Docker environment on <a href="#">macOS</a>, <a href="#">Windows</a>, and <a href="#">Linux</a>. For a primer on Docker and container basics, see the <a href="#">Docker overview</a>.</p> <p>Docker must be configured to allow the containers to connect with and send billing data to Azure.</p> <p><b>On Windows</b>, Docker must also be configured to support Linux containers.</p>
Familiarity with Docker	<p>You should have a basic understanding of Docker concepts, like registries, repositories, containers, and container images, as well as knowledge of basic <code>docker</code> commands.</p>
Speech resource	<p>In order to use these containers, you must have:</p> <p>An Azure <i>Speech</i> resource to get the associated API key and endpoint URI. Both values are available on the Azure portal's <b>Speech</b> Overview and Keys pages. They are both required to start the container.</p> <p><b>{API_KEY}</b>: One of the two available resource keys on the <a href="#">Keys</a> page</p> <p><b>{ENDPOINT_URI}</b>: The endpoint as provided on the <a href="#">Overview</a> page</p>

## Request access to the container registry

Fill out and submit the [Cognitive Services Speech Containers Request form](#) to request access to the container.

The form requests information about you, your company, and the user scenario for which you'll use the container. After you've submitted the form, the Azure Cognitive Services team reviews it to ensure that you meet the criteria for access to the private container registry.

### IMPORTANT

You must use an email address that's associated with either a Microsoft Account (MSA) or Azure Active Directory (Azure AD) account in the form.

If your request is approved, you'll receive an email with instructions that describe how to obtain your credentials and access the private container registry.

## Use the Docker CLI to authenticate the private container registry

You can authenticate with the private container registry for Cognitive Services Containers in any of several ways, but the recommended method from the command line is to use the [Docker CLI](#).

Use the `docker login` command, as shown in the following example, to log in to `containerpreview.azurecr.io`, the private container registry for Cognitive Services Containers. Replace `<username>` with the user name and `<password>` with the password that's provided in the credentials you received from the Azure Cognitive Services team.

```
docker login containerpreview.azurecr.io -u <username> -p <password>
```

If you've secured your credentials in a text file, you can concatenate the contents of that text file, by using the `cat` command, to the `docker login` command, as shown in the following example. Replace `<passwordFile>` with the path and name of the text file that contains the password and `<username>` with the user name that's provided in your credentials.

```
cat <passwordFile> | docker login containerpreview.azurecr.io -u <username> --password-stdin
```

## Gathering required parameters

There are three primary parameters for all Cognitive Services' containers that are required. The end-user license agreement (EULA) must be present with a value of `accept`. Additionally, both an Endpoint URL and API Key are needed.

### Endpoint URI `{ENDPOINT_URI}`

The **Endpoint** URI value is available on the Azure portal *Overview* page of the corresponding Cognitive Service resource. Navigate to the *Overview* page, hover over the Endpoint, and a `Copy to clipboard` icon will appear. Copy and use where needed.

Home > widgets

**widgets**  
Cognitive Services

Search (Ctrl+ /)

Unavailable setting Delete

Overview (highlighted with a red arrow)

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

RESOURCE MANAGEMENT

- Keys (highlighted with a red arrow)
- Quick start
- Pricing tier
- Billing By Subscription

Resource group (change)  
**widgets-resource-group**

Status  
Active

Location  
North Central US

Subscription (change)  
**widgets-subscription**

Subscription ID  
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx

Tags (change)  
Click here to add tags

API type  
<API Type>

Pricing tier  
Standard

Endpoint  
**https://widgets.cognitiveservices.azure.com/api/example-endpoint**

Manage keys  
Show access keys ...

Copy to clipboard (highlighted with a red box)

### Keys `{API_KEY}`

This key is used to start the container, and is available on the Azure portal's Keys page of the corresponding Cognitive Service resource. Navigate to the *Keys* page, and click on the `Copy to clipboard` icon.

Home > widgets - Keys

**widgets - Keys**  
Cognitive Services

Search (Ctrl+ /)

Regenerate Key1 Regenerate Key2

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

RESOURCE MANAGEMENT

Keys (highlighted with a red arrow)

Quick start

Pricing tier

Billing By Subscription

NAME  
widgets

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

KEY 1  
<key 1 value> (highlighted with a red box)

KEY 2  
<key 2 value>

## IMPORTANT

These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely, for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

## The host computer

The host is a x64-based computer that runs the Docker container. It can be a computer on your premises or a Docker hosting service in Azure, such as:

- [Azure Kubernetes Service](#).
- [Azure Container Instances](#).
- A [Kubernetes cluster deployed to Azure Stack](#). For more information, see [Deploy Kubernetes to Azure Stack](#).

### Advanced Vector Extension support

The **host** is the computer that runs the docker container. The host *must support* [Advanced Vector Extensions](#) (AVX2). You can check for AVX2 support on Linux hosts with the following command:

```
grep -q avx2 /proc/cpuinfo && echo AVX2 supported || echo No AVX2 support detected
```

## WARNING

The host computer is *required* to support AVX2. The container *will not* function correctly without AVX2 support.

### Container requirements and recommendations

The following table describes the minimum and recommended allocation of resources for each Speech container.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

CONTAINER	MINIMUM	RECOMMENDED
Speech-to-text	2 core, 2-GB memory	4 core, 4-GB memory

- Each core must be at least 2.6 gigahertz (GHz) or faster.

Core and memory correspond to the `--cpus` and `--memory` settings, which are used as part of the `docker run` command.

## NOTE

The minimum and recommended are based off of Docker limits, *not* the host machine resources. For example, speech-to-text containers memory map portions of a large language model, and it is *recommended* that the entire file fits in memory, which is an additional 4-6 GB. Also, the first run of either container may take longer, since models are being paged into memory.

Get the container image with `docker pull`

Container images for Speech are available in the following Container Registry.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

CONTAINER	REPOSITORY
Speech-to-text	<code>containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text:latest</code>

#### TIP

You can use the `docker images` command to list your downloaded container images. For example, the following command lists the ID, repository, and tag of each downloaded container image, formatted as a table:

```
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
```

IMAGE ID	REPOSITORY	TAG
<image-id>	<repository-path/name>	<tag-name>

### Docker pull for the Speech containers

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

#### Docker pull for the Speech-to-text container

Use the `docker pull` command to download a container image from Container Preview registry.

```
docker pull containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text:latest
```

#### IMPORTANT

The `latest` tag pulls the `en-us` locale. For additional locales see [Speech-to-text locales](#).

#### Speech-to-text locales

All tags, except for `latest` are in the following format and are case-sensitive:

```
<major>.<minor>.<patch>-<platform>-<locale>-<prerelease>
```

The following tag is an example of the format:

```
2.0.0-amd64-en-us-preview
```

For all of the supported locales of the **speech-to-text** container, please see [Speech-to-text image tags](#).

## How to use the container

Once the container is on the [host computer](#), use the following process to work with the container.

1. Run the container, with the required billing settings. More [examples](#) of the `docker run` command are available.
2. Query the container's prediction endpoint.

## Run the container with `docker run`

Use the `docker run` command to run the container. Refer to [gathering required parameters](#) for details on how to get the `{Endpoint_URI}` and `{API_Key}` values. Additional [examples](#) of the `docker run` command are also available.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

To run the *Speech-to-text* container, execute the following `docker run` command.

```
docker run --rm -it -p 5000:5000 --memory 4g --cpus 4 \
containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

This command:

- Runs a *Speech-to-text* container from the container image.
- Allocates 4 CPU cores and 4 gigabytes (GB) of memory.
- Exposes TCP port 5000 and allocates a pseudo-TTY for the container.
- Automatically removes the container after it exits. The container image is still available on the host computer.

### IMPORTANT

The `Eula`, `Billing`, and `ApiKey` options must be specified to run the container; otherwise, the container won't start. For more information, see [Billing](#).

## Query the container's prediction endpoint

CONTAINERS	SDK HOST URL	PROTOCOL
Speech-to-text and Custom Speech-to-text	<code>ws://localhost:5000</code>	WS
Text-to-speech and Custom Text-to-speech	<code>http://localhost:5000</code>	HTTP

For more information on using WSS and HTTPS protocols, see [container security](#).

### Speech-to-text or Custom Speech-to-text

The container provides websocket-based query endpoint APIs, that are accessed through the [Speech SDK](#). By default, the Speech SDK uses online speech services. To use the container, you need to change the initialization method.

## TIP

When using the Speech SDK with containers, you do not need to provide the Azure Speech resource [subscription key or an authentication bearer token](#).

See the examples below.

- [C#](#)
- [Python](#)

Change from using this Azure-cloud initialization call:

```
var config = SpeechConfig.FromSubscription("YourSubscriptionKey", "YourServiceRegion");
```

to this call using the container [host](#):

```
var config = SpeechConfig.FromHost(
 new Uri("ws://localhost:5000"));
```

## Text-to-speech or Custom Text-to-speech

The container provides [REST-based endpoint APIs](#). There are many [sample source code projects](#) for platform, framework, and language variations available.

With the *Standard Text-to-speech* container, you should rely on the locale and voice of the image tag you downloaded. For example, if you downloaded the `latest` tag the default locale is `en-US` and the `JessaRUS` voice. The `{VOICE_NAME}` argument would then be `en-US-JessaRUS`. See the example SSML below:

```
<speak version="1.0" xml:lang="en-US">
 <voice name="en-US-JessaRUS">
 This text will get converted into synthesized speech.
 </voice>
</speak>
```

However, for *Custom Text-to-speech* you'll need to obtain the **Voice / model** from the [custom voice portal](#). The custom model name is synonymous with the voice name. Navigate to the **Training** page, and copy the **Voice / model** to use as the `{VOICE_NAME}` argument.

The screenshot shows the Azure Cognitive Services | Speech Studio interface. The top navigation bar includes links for Cognitive Services, Speech Studio, Custom Voice, widgets-custom-voice, Training, and a dropdown menu. On the right side of the header are icons for search, settings, help, and a JS button.

The main content area displays a 'Widgets custom voice model' card. It shows the following details:

Gender	Language	ModelType	Utterances	Status	Model ID
Male	English (United States)	Statistical parametric	2	Succeeded	XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX

Below the card, there are buttons for 'Add test' and file operations (New, Open, Delete). A 'Tests' section lists a single entry:

Text input	Voice / model	Created	Status	Audio
Hi, this is my custom voice.	custom-voice-model	10/15/2019 7:29 AM	Succeeded	(play, download icons)

See the example SSML below:

```

<speak version="1.0" xml:lang="en-US">
 <voice name="custom-voice-model">
 This text will get converted into synthesized speech.
 </voice>
</speak>

```

Let's construct an HTTP POST request, providing a few headers and a data payload. Replace the `{VOICE_NAME}` placeholder with your own value.

```

curl -s -v -X POST http://localhost:5000/speech/synthesize/cognitiveservices/v1 \
-H 'Accept: audio/*' \
-H 'Content-Type: application/ssml+xml' \
-H 'X-Microsoft-OutputFormat: riff-16khz-16bit-mono-pcm' \
-d '<speak version="1.0" xml:lang="en-US"><voice name="{VOICE_NAME}">This is a test, only a test.</voice>
</speak>''

```

This command:

- Constructs an HTTP POST request for the `speech/synthesize/cognitiveservices/v1` endpoint.
- Specifies an `Accept` header of `audio/*`
- Specifies a `Content-Type` header of `application/ssml+xml`, for more information, see [request body](#).
- Specifies a `X-Microsoft-OutputFormat` header of `riff-16khz-16bit-mono-pcm`, for more options see [audio output](#).
- Sends the [Speech Synthesis Markup Language \(SSML\)](#) request given the `{VOICE_NAME}` to the endpoint.

#### Run multiple containers on the same host

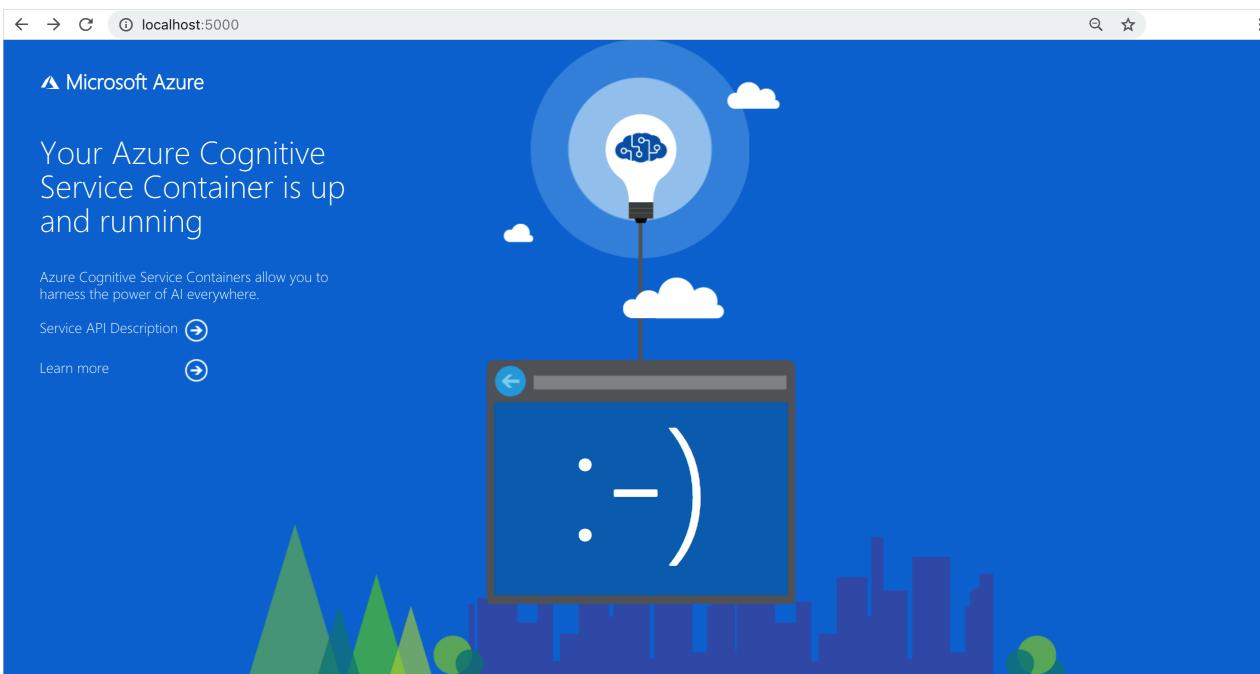
If you intend to run multiple containers with exposed ports, make sure to run each container with a different exposed port. For example, run the first container on port 5000 and the second container on port 5001.

You can have this container and a different Azure Cognitive Services container running on the HOST together. You also can have multiple containers of the same Cognitive Services container running.

## Validate that a container is running

There are several ways to validate that the container is running. Locate the *External IP* address and exposed port of the container in question, and open your favorite web browser. Use the various request URLs below to validate the container is running. The example request URLs listed below are `http://localhost:5000`, but your specific container may vary. Keep in mind that you're to rely on your container's *External IP* address and exposed port.

REQUEST URL	PURPOSE
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/status</code>	Requested with an HTTP GET, to validate that the container is running without causing an endpoint query. This request can be used for Kubernetes <a href="#">liveness and readiness probes</a> .
<code>http://localhost:5000/swagger</code>	The container provides a full set of documentation for the endpoints and a <b>Try it out</b> feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.



## Stop the container

To shut down the container, in the command-line environment where the container is running, select **Ctrl+C**.

## Troubleshooting

When starting or running the container, you may experience issues. Use an output **mount** and enable logging. Doing so will allow the container to generate log files that are helpful when troubleshooting issues.

### TIP

For more troubleshooting information and guidance, see [Cognitive Services containers frequently asked questions \(FAQ\)](#).

## Billing

The Speech containers send billing information to Azure, using a *Speech* resource on your Azure account.

Queries to the container are billed at the pricing tier of the Azure resource that's used for the <ApiKey> .

Azure Cognitive Services containers aren't licensed to run without being connected to the billing endpoint for metering. You must enable the containers to communicate billing information with the billing endpoint at all times. Cognitive Services containers don't send customer data, such as the image or text that's being analyzed, to Microsoft.

### Connect to Azure

The container needs the billing argument values to run. These values allow the container to connect to the billing endpoint. The container reports usage about every 10 to 15 minutes. If the container doesn't connect to Azure within the allowed time window, the container continues to run but doesn't serve queries until the billing endpoint is restored. The connection is attempted 10 times at the same time interval of 10 to 15 minutes. If it can't connect to the billing endpoint within the 10 tries, the container stops running.

### Billing arguments

For the `docker run` command to start the container, all three of the following options must be specified with valid values:

OPTION	DESCRIPTION
ApiKey	The API key of the Cognitive Services resource that's used to track billing information. The value of this option must be set to an API key for the provisioned resource that's specified in <code>Billing</code> .
Billing	The endpoint of the Cognitive Services resource that's used to track billing information. The value of this option must be set to the endpoint URI of a provisioned Azure resource.
Eula	Indicates that you accepted the license for the container. The value of this option must be set to <code>accept</code> .

For more information about these options, see [Configure containers](#).

## Blog posts

- [Running Cognitive Services Containers](#)
- [Azure Cognitive Services](#)

## Developer samples

Developer samples are available at our [GitHub repository](#).

## View webinar

Join the [webinar](#) to learn about:

- How to deploy Cognitive Services to any machine using Docker
- How to deploy Cognitive Services to AKS

## Summary

In this article, you learned concepts and workflow for downloading, installing, and running Speech containers. In summary:

- Speech provides four Linux containers for Docker, encapsulating various capabilities:
  - *Speech-to-text*
  - *Custom Speech-to-text*
  - *Text-to-speech*
  - *Custom Text-to-speech*
- Container images are downloaded from the container registry in Azure.
- Container images run in Docker.
- You can use either the REST API or SDK to call operations in Speech containers by specifying the host URI of the container.
- You're required to provide billing information when instantiating a container.

**IMPORTANT**

Cognitive Services containers are not licensed to run without being connected to Azure for metering. Customers need to enable the containers to communicate billing information with the metering service at all times. Cognitive Services containers do not send customer data (e.g., the image or text that is being analyzed) to Microsoft.

## Next steps

- Review [configure containers](#) for configuration settings
- Learn how to use [Speech service containers with Kubernetes and Helm](#)
- Use more [Cognitive Services containers](#)

# Configure Speech service containers

12/4/2019 • 9 minutes to read • [Edit Online](#)

Speech containers enable customers to build one speech application architecture that is optimized to take advantage of both robust cloud capabilities and edge locality. The four speech containers we support now are, **speech-to-text**, **custom-speech-to-text**, **text-to-speech**, and **custom-text-to-speech**.

The **Speech** container runtime environment is configured using the `docker run` command arguments. This container has several required settings, along with a few optional settings. Several [examples](#) of the command are available. The container-specific settings are the billing settings.

## Configuration settings

The container has the following configuration settings:

REQUIRED	SETTING	PURPOSE
Yes	<a href="#">ApiKey</a>	Tracks billing information.
No	<a href="#">ApplicationInsights</a>	Enables adding <a href="#">Azure Application Insights</a> telemetry support to your container.
Yes	<a href="#">Billing</a>	Specifies the endpoint URI of the service resource on Azure.
Yes	<a href="#">Eula</a>	Indicates that you've accepted the license for the container.
No	<a href="#">Fluentd</a>	Writes log and, optionally, metric data to a Fluentd server.
No	<a href="#">HTTP Proxy</a>	Configures an HTTP proxy for making outbound requests.
No	<a href="#">Logging</a>	Provides ASP.NET Core logging support for your container.
No	<a href="#">Mounts</a>	Reads and writes data from the host computer to the container and from the container back to the host computer.

### IMPORTANT

The [ApiKey](#), [Billing](#), and [Eula](#) settings are used together, and you must provide valid values for all three of them; otherwise your container won't start. For more information about using these configuration settings to instantiate a container, see [Billing](#).

## ApiKey configuration setting

The [ApiKey](#) setting specifies the Azure resource key used to track billing information for the container. You must specify a value for the ApiKey and the value must be a valid key for the *Speech* resource specified for the [Billing](#) configuration setting.

This setting can be found in the following place:

- Azure portal: **Speech's** Resource Management, under **Keys**

## ApplicationInsights setting

The `ApplicationInsights` setting allows you to add [Azure Application Insights](#) telemetry support to your container.

Application Insights provides in-depth monitoring of your container. You can easily monitor your container for availability, performance, and usage. You can also quickly identify and diagnose errors in your container.

The following table describes the configuration settings supported under the `ApplicationInsights` section.

REQUIRED	NAME	DATA TYPE	DESCRIPTION
No	<code>InstrumentationKey</code>	String	<p>The instrumentation key of the Application Insights instance to which telemetry data for the container is sent. For more information, see <a href="#">Application Insights for ASP.NET Core</a>.</p> <p>Example:  <code>InstrumentationKey=123456789</code></p>

## Billing configuration setting

The `Billing` setting specifies the endpoint URI of the *Speech* resource on Azure used to meter billing information for the container. You must specify a value for this configuration setting, and the value must be a valid endpoint URI for a *Speech* resource on Azure. The container reports usage about every 10 to 15 minutes.

This setting can be found in the following place:

- Azure portal: **Speech's** Overview, labeled `Endpoint`

REQUIRED	NAME	DATA TYPE	DESCRIPTION
Yes	<code>Billing</code>	String	<p>Billing endpoint URI. For more information on obtaining the billing URI, see <a href="#">gathering required parameters</a>. For more information and a complete list of regional endpoints, see <a href="#">Custom subdomain names for Cognitive Services</a>.</p>

## Eula setting

The `Eula` setting indicates that you've accepted the license for the container. You must specify a value for this configuration setting, and the value must be set to `accept`.

REQUIRED	NAME	DATA TYPE	DESCRIPTION
Yes	<code>Eula</code>	String	<p>License acceptance</p> <p>Example:  <code>Eula=accept</code></p>

Cognitive Services containers are licensed under [your agreement](#) governing your use of Azure. If you do not have an existing agreement governing your use of Azure, you agree that your agreement governing use of Azure is the [Microsoft Online Subscription Agreement](#), which incorporates the [Online Services Terms](#). For previews, you also agree to the [Supplemental Terms of Use for Microsoft Azure Previews](#). By using the container you agree to these terms.

## Fluentd settings

Fluentd is an open-source data collector for unified logging. The `Fluentd` settings manage the container's connection to a [Fluentd](#) server. The container includes a Fluentd logging provider, which allows your container to write logs and, optionally, metric data to a Fluentd server.

The following table describes the configuration settings supported under the `Fluentd` section.

NAME	DATA TYPE	DESCRIPTION
<code>Host</code>	String	The IP address or DNS host name of the Fluentd server.
<code>Port</code>	Integer	The port of the Fluentd server. The default value is 24224.
<code>HeartbeatMs</code>	Integer	The heartbeat interval, in milliseconds. If no event traffic has been sent before this interval expires, a heartbeat is sent to the Fluentd server. The default value is 60000 milliseconds (1 minute).
<code>SendBufferSize</code>	Integer	The network buffer space, in bytes, allocated for send operations. The default value is 32768 bytes (32 kilobytes).
<code>TlsConnectionEstablishmentTimeoutMs</code>	Integer	The timeout, in milliseconds, to establish a SSL/TLS connection with the Fluentd server. The default value is 10000 milliseconds (10 seconds). If <code>UseTLS</code> is set to false, this value is ignored.
<code>UseTLS</code>	Boolean	Indicates whether the container should use SSL/TLS for communicating with the Fluentd server. The default value is false.

## HTTP proxy credentials settings

If you need to configure an HTTP proxy for making outbound requests, use these two arguments:

NAME	DATA TYPE	DESCRIPTION
<code>HTTP_PROXY</code>	string	The proxy to use, for example, <code>http://proxy:8888</code> <code>&lt;proxy-url&gt;</code>
<code>HTTP_PROXY_CREDS</code>	string	Any credentials needed to authenticate against the proxy, for example, <code>username:password</code> .
<code>&lt;proxy-user&gt;</code>	string	The user for the proxy.
<code>&lt;proxy-password&gt;</code>	string	The password associated with <code>&lt;proxy-user&gt;</code> for the proxy.

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
HTTP_PROXY=<proxy-url> \
HTTP_PROXY_CREDS=<proxy-user>:<proxy-password> \
```

## Logging settings

The `Logging` settings manage ASP.NET Core logging support for your container. You can use the same configuration settings and values for your container that you use for an ASP.NET Core application.

The following logging providers are supported by the container:

Provider	Purpose
Console	The ASP.NET Core <code>Console</code> logging provider. All of the ASP.NET Core configuration settings and default values for this logging provider are supported.
Debug	The ASP.NET Core <code>Debug</code> logging provider. All of the ASP.NET Core configuration settings and default values for this logging provider are supported.
Disk	The JSON logging provider. This logging provider writes log data to the output mount.

This container command stores logging information in the JSON format to the output mount:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Disk:Format=json
```

This container command shows debugging information, prefixed with `dbug`, while the container is running:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Console:LogLevel:Default=Debug
```

### Disk logging

The `Disk` logging provider supports the following configuration settings:

Name	Data Type	Description

NAME	DATA TYPE	DESCRIPTION
Format	String	The output format for log files. <b>Note:</b> This value must be set to <code>json</code> to enable the logging provider. If this value is specified without also specifying an output mount while instantiating a container, an error occurs.
MaxFileSize	Integer	The maximum size, in megabytes (MB), of a log file. When the size of the current log file meets or exceeds this value, a new log file is started by the logging provider. If -1 is specified, the size of the log file is limited only by the maximum file size, if any, for the output mount. The default value is 1.

For more information about configuring ASP.NET Core logging support, see [Settings file configuration](#).

## Mount settings

Use bind mounts to read and write data to and from the container. You can specify an input mount or output mount by specifying the `--mount` option in the [docker run](#) command.

The Standard Speech containers don't use input or output mounts to store training or service data. However, custom speech containers rely on volume mounts.

The exact syntax of the host mount location varies depending on the host operating system. Additionally, the [host computer](#)'s mount location may not be accessible due to a conflict between permissions used by the docker service account and the host mount location permissions.

OPTIONAL	NAME	DATA TYPE	DESCRIPTION
Not allowed	Input	String	Standard Speech containers do not use this. Custom speech containers use <a href="#">volume mounts</a> .
Optional	Output	String	The target of the output mount. The default value is <code>/output</code> . This is the location of the logs. This includes container logs.

Example:

```
--mount
type=bind,src=c:\output,target=/o
```

## Volume mount settings

The custom speech containers use [volume mounts](#) to persist custom models. You can specify a volume mount by adding the `-v` (or `--volume`) option to the [docker run](#) command.

Custom models are downloaded the first time that a new model is ingested as part of the custom speech container docker run command. Sequential runs of the same `ModelId` for a custom speech container will use the previously downloaded model. If the volume mount is not provided, custom models cannot be persisted.

The volume mount setting consists of three color `:` separated fields:

1. The first field is the name of the volume on the host machine, for example `C:\input`.

2. The second field is the directory in the container, for example `/usr/local/models`.
3. The third field (optional) is a comma-separated list of options, for more information see [use volumes](#).

### Volume mount example

```
-v C:\input:/usr/local/models
```

This command mounts the host machine `C:\input` directory to the containers `/usr/local/models` directory.

#### IMPORTANT

The volume mount settings are only applicable to **Custom Speech-to-text** and **Custom Text-to-speech** containers. The standard **Speech-to-text** and **Text-to-speech** containers do not use volume mounts.

## Example docker run commands

The following examples use the configuration settings to illustrate how to write and use `docker run` commands. Once running, the container continues to run until you [stop](#) it.

- **Line-continuation character:** The Docker commands in the following sections use the back slash, `\`, as a line continuation character. Replace or remove this based on your host operating system's requirements.
- **Argument order:** Do not change the order of the arguments unless you are familiar with Docker containers.

Replace `{argument_name}` with your own values:

PLACEHOLDER	VALUE	FORMAT OR EXAMPLE
<code>{API_KEY}</code>	The endpoint key of the <code>Speech</code> resource on the Azure <code>Speech</code> Keys page.	<code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code>
<code>{ENDPOINT_URI}</code>	The billing endpoint value is available on the Azure <code>Speech</code> Overview page.	See <a href="#">gathering required parameters</a> for explicit examples.

#### NOTE

New resources created after July 1, 2019, will use custom subdomain names. For more information and a complete list of regional endpoints, see [Custom subdomain names for Cognitive Services](#).

#### IMPORTANT

The `Eula`, `Billing`, and `ApiKey` options must be specified to run the container; otherwise, the container won't start. For more information, see [Billing](#). The `ApiKey` value is the **Key** from the Azure Speech Resource keys page.

## Speech container Docker examples

The following Docker examples are for the Speech container.

- [Speech-to-text](#)
- [Custom Speech-to-text](#)
- [Text-to-speech](#)
- [Custom Text-to-speech](#)

### Basic example for Speech-to-text

```
docker run --rm -it -p 5000:5000 --memory 4g --cpus 4 \
containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

## Logging example for Speech-to-text

```
docker run --rm -it -p 5000:5000 --memory 4g --cpus 4 \
containerpreview.azurecr.io/microsoft/cognitive-services-custom-speech-to-text \
Eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY} \
Logging:Console:LogLevel:Default=Information
```

## Next steps

- Review [How to install and run containers](#)

# Use Speech service containers with Kubernetes and Helm

12/4/2019 • 11 minutes to read • [Edit Online](#)

One option to manage your Speech containers on-premises is to use Kubernetes and Helm. Using Kubernetes and Helm to define the speech-to-text and text-to-speech container images, we'll create a Kubernetes package. This package will be deployed to a Kubernetes cluster on-premises. Finally, we'll explore how to test the deployed services and various configuration options. For more information about running Docker containers without Kubernetes orchestration, see [install and run Speech service containers](#).

## Prerequisites

The following prerequisites before using Speech containers on-premises:

REQUIRED	PURPOSE
Azure Account	If you don't have an Azure subscription, create a <a href="#">free account</a> before you begin.
Container Registry access	In order for Kubernetes to pull the docker images into the cluster, it will need access to the container registry.
Kubernetes CLI	The <a href="#">Kubernetes CLI</a> is required for managing the shared credentials from the container registry. Kubernetes is also needed before Helm, which is the Kubernetes package manager.
Helm CLI	As part of the <a href="#">Helm CLI</a> install, you'll also need to initialize Helm, which will install <a href="#">Tiller</a> .
Speech resource	In order to use these containers, you must have:  A <i>Speech</i> Azure resource to get the associated billing key and billing endpoint URI. Both values are available on the Azure portal's <b>Speech</b> Overview and Keys pages and are required to start the container.  <b>{API_KEY}</b> : resource key  <b>{ENDPOINT_URI}</b> : endpoint URI example is: <code>https://westus.api.cognitive.microsoft.com/sts/v1.0</code>

## The recommended host computer configuration

Refer to the [Speech service container host computer](#) details as a reference. This *helm chart* automatically calculates CPU and memory requirements based on how many decodes (concurrent requests) that the user specifies. Additionally, it will adjust based on whether optimizations for audio/text input are configured as `enabled`. The helm chart defaults to, two concurrent requests and disabling optimization.

Service	CPU / Container	Memory / Container
<b>Speech-to-Text</b>	one decoder requires a minimum of 1,150 millicores. If the <code>optimizedForAudioFile</code> is enabled, then 1,950 millicores are required. (default: two decoders)	Required: 2 GB Limited: 4 GB
<b>Text-to-Speech</b>	one concurrent request requires a minimum of 500 millicores. If the <code>optimizeForTurboMode</code> is enabled, then 1,000 millicores are required. (default: two concurrent requests)	Required: 1 GB Limited: 2 GB

## Connect to the Kubernetes cluster

The host computer is expected to have an available Kubernetes cluster. See this tutorial on [deploying a Kubernetes cluster](#) for a conceptual understanding of how to deploy a Kubernetes cluster to a host computer.

### Sharing Docker credentials with the Kubernetes cluster

To allow the Kubernetes cluster to `docker pull` the configured image(s) from the `mcr.microsoft.com` container registry, you need to transfer the docker credentials into the cluster. Execute the `kubectl create` command below to create a *docker-registry secret* based on the credentials provided from the container registry access prerequisite.

From your command-line interface of choice, run the following command. Be sure to replace the `<username>`, `<password>`, and `<email-address>` with the container registry credentials.

```
kubectl create secret docker-registry mcr \
--docker-server=mcr.microsoft.com \
--docker-username=<username> \
--docker-password=<password> \
--docker-email=<email-address>
```

#### NOTE

If you already have access to the `mcr.microsoft.com` container registry, you could create a Kubernetes secret using the generic flag instead. Consider the following command that executes against your Docker configuration JSON.

```
kubectl create secret generic mcr \
--from-file=.dockerconfigjson=~/docker/config.json \
--type=kubernetes.io/dockerconfigjson
```

The following output is printed to the console when the secret has been successfully created.

```
secret "mcr" created
```

To verify that the secret has been created, execute the `kubectl get` with the `secrets` flag.

```
kubectl get secrets
```

Executing the `kubectl get secrets` prints all the configured secrets.

NAME	TYPE	DATA	AGE
mcr	kubernetes.io/dockerconfigjson	1	30s

## Configure Helm chart values for deployment

Visit the [Microsoft Helm Hub](#) for all the publicly available helm charts offered by Microsoft. From the Microsoft Helm Hub, you'll find the **Cognitive Services Speech On-Premises Chart**. The **Cognitive Services Speech On-Premises** is the chart we'll install, but we must first create an `config-values.yaml` file with explicit configurations. Let's start by adding the Microsoft repository to our Helm instance.

```
helm repo add microsoft https://microsoft.github.io/charts/repo
```

Next, we'll configure our Helm chart values. Copy and paste the following YAML into a file named `config-values.yaml`. For more information on customizing the **Cognitive Services Speech On-Premises Helm Chart**, see [customize helm charts](#). Replace the `# {ENDPOINT_URI}` and `# {API_KEY}` comments with your own values.

```
These settings are deployment specific and users can provide customizations

speech-to-text configurations
speechToText:
 enabled: true
 numberOfWorkers: 3
 optimizeForAudioFile: true
 image:
 registry: mcr.microsoft.com
 repository: azure-cognitive-services/speech-to-text
 tag: latest
 pullSecrets:
 - mcr # Or an existing secret
 args:
 eula: accept
 billing: # {ENDPOINT_URI}
 apikey: # {API_KEY}

text-to-speech configurations
textToSpeech:
 enabled: true
 numberOfWorkers: 3
 optimizeForTurboMode: true
 image:
 registry: mcr.microsoft.com
 repository: azure-cognitive-services/text-to-speech
 tag: latest
 pullSecrets:
 - mcr # Or an existing secret
 args:
 eula: accept
 billing: # {ENDPOINT_URI}
 apikey: # {API_KEY}
```

### IMPORTANT

If the `billing` and `apikey` values are not provided, the services will expire after 15 min. Likewise, verification will fail as the services will not be available.

## The Kubernetes package (Helm chart)

The *Helm chart* contains the configuration of which docker image(s) to pull from the `mcr.microsoft.com` container registry.

A [Helm chart](#) is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on.

The provided *Helm charts* pull the docker images of the Speech service, both text-to-speech and the speech-to-text services from the `mcr.microsoft.com` container registry.

## Install the Helm chart on the Kubernetes cluster

To install the *helm chart* we'll need to execute the `helm install` command, replacing the `<config-values.yaml>` with the appropriate path and file name argument. The `microsoft/cognitive-services-speech-onpremise` Helm chart referenced below is available on the [Microsoft Helm Hub here](#).

```
helm install microsoft/cognitive-services-speech-onpremise \
--version 0.1.1 \
--values <config-values.yaml> \
--name onprem-speech
```

Here is an example output you might expect to see from a successful install execution:

```
NAME: onprem-speech
LAST DEPLOYED: Tue Jul 2 12:51:42 2019
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME READY STATUS RESTARTS AGE
speech-to-text-7664f5f465-87w2d 0/1 Pending 0 0s
speech-to-text-7664f5f465-klbr8 0/1 ContainerCreating 0 0s
text-to-speech-56f8fb685b-4jtzh 0/1 ContainerCreating 0 0s
text-to-speech-56f8fb685b-frwxf 0/1 Pending 0 0s

==> v1/Service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
speech-to-text LoadBalancer 10.0.252.106 <pending> 80:31811/TCP 1s
text-to-speech LoadBalancer 10.0.125.187 <pending> 80:31247/TCP 0s

==> v1beta1/PodDisruptionBudget
NAME MIN AVAILABLE MAX UNAVAILABLE ALLOWED DISRUPTIONS AGE
speech-to-text-poddisruptionbudget N/A 20% 0 1s
text-to-speech-poddisruptionbudget N/A 20% 0 1s

==> v1beta2/Deployment
NAME READY UP-TO-DATE AVAILABLE AGE
speech-to-text 0/2 2 0 0s
text-to-speech 0/2 2 0 0s

==> v2beta2/HorizontalPodAutoscaler
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
speech-to-text-autoscaler Deployment/speech-to-text <unknown>/50% 2 10 0 0s
text-to-speech-autoscaler Deployment/text-to-speech <unknown>/50% 2 10 0 0s

NOTES:
cognitive-services-speech-onpremise has been installed!
Release is named onprem-speech
```

The Kubernetes deployment can take over several minutes to complete. To confirm that both pods and services are properly deployed and available, execute the following command:

```
kubectl get all
```

You should expect to see something similar to the following output:

```
NAME READY STATUS RESTARTS AGE
pod/speech-to-text-7664f5f465-87w2d 1/1 Running 0 34m
pod/speech-to-text-7664f5f465-klbr8 1/1 Running 0 34m
pod/text-to-speech-56f8fb685b-4jtzh 1/1 Running 0 34m
pod/text-to-speech-56f8fb685b-frwxf 1/1 Running 0 34m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/kubernetes ClusterIP 10.0.0.1 <none> 443/TCP 3h
service/speech-to-text LoadBalancer 10.0.252.106 52.162.123.151 80:31811/TCP 34m
service/text-to-speech LoadBalancer 10.0.125.187 65.52.233.162 80:31247/TCP 34m

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
deployment.apps/speech-to-text 2 2 2 2 34m
deployment.apps/text-to-speech 2 2 2 2 34m

NAME DESIRED CURRENT READY AGE
replicaset.apps/speech-to-text-7664f5f465 2 2 2 34m
replicaset.apps/text-to-speech-56f8fb685b 2 2 2 34m

NAME REFERENCE TARGETS MINPODS
MAXPODS REPLICAS AGE
horizontalpodautoscaler.autoscaling/speech-to-text-autoscaler Deployment/speech-to-text 1%/50% 2
10 2 34m
horizontalpodautoscaler.autoscaling/text-to-speech-autoscaler Deployment/text-to-speech 0%/50% 2
10 2 34m
```

## Verify Helm deployment with Helm tests

The installed Helm charts define *Helm tests*, which serve as a convenience for verification. These tests validate service readiness. To verify both **speech-to-text** and **text-to-speech** services, we'll execute the [Helm test](#) command.

```
helm test onprem-speech
```

### IMPORTANT

These tests will fail if the POD status is not `Running` or if the deployment is not listed under the `AVAILABLE` column. Be patient as this can take over ten minutes to complete.

These tests will output various status results:

```
RUNNING: speech-to-text-readiness-test
PASSED: speech-to-text-readiness-test
RUNNING: text-to-speech-readiness-test
PASSED: text-to-speech-readiness-test
```

As an alternative to executing the *helm tests*, you could collect the *External IP* addresses and corresponding ports from the `kubectl get all` command. Using the IP and port, open a web browser and navigate to

`http://<external-ip>:<port>/swagger/index.html` to view the API swagger page(s).

# Customize Helm charts

Helm charts are hierarchical. Being hierarchical allows for chart inheritance, it also caters to the concept of specificity, where settings that are more specific override inherited rules.

## Speech (umbrella chart)

Values in the top-level "umbrella" chart override the corresponding sub-chart values. Therefore, all on-premises customized values should be added here.

PARAMETER	DESCRIPTION	DEFAULT
<code>speechToText.enabled</code>	Whether the <b>speech-to-text</b> service is enabled.	<code>true</code>
<code>speechToText.verification.enabled</code>	Whether the <code>helm test</code> capability for <b>speech-to-text</b> service is enabled.	<code>true</code>
<code>speechToText.verification.image.registry</code>	The docker image repository that <code>helm test</code> uses to test <b>speech-to-text</b> service. Helm creates separate pod inside the cluster for testing and pulls the <i>test-use</i> image from this registry.	<code>docker.io</code>
<code>speechToText.verification.image.repository</code>	The docker image repository that <code>helm test</code> uses to test <b>speech-to-text</b> service. Helm test pod uses this repository to pull <i>test-use</i> image.	<code>antsu/on-prem-client</code>
<code>speechToText.verification.image.tag</code>	The docker image tag used with <code>helm test</code> for <b>speech-to-text</b> service. Helm test pod uses this tag to pull <i>test-use</i> image.	<code>latest</code>
<code>speechToText.verification.image.pullByHash</code>	Whether the <i>test-use</i> docker image is pulled by hash. If <code>true</code> , <code>speechToText.verification.image.hash</code> should be added, with valid image hash value.	<code>false</code>
<code>speechToText.verification.image.arguments</code>	The arguments used to execute the <i>test-use</i> docker image. Helm test pod passes these arguments to the container when running <code>helm test</code> .	<code>./speech-to-text-client ./audio/whatstheweatherlike.wav --expect=What's the weather like? --host=\$(SPEECH_TO_TEXT_HOST) --port=\$(SPEECH_TO_TEXT_PORT)</code>
<code>textToSpeech.enabled</code>	Whether the <b>text-to-speech</b> service is enabled.	<code>true</code>
<code>textToSpeech.verification.enabled</code>	Whether the <code>helm test</code> capability for <b>speech-to-text</b> service is enabled.	<code>true</code>
<code>textToSpeech.verification.image.registry</code>	The docker image repository that <code>helm test</code> uses to test <b>speech-to-text</b> service. Helm creates separate pod inside the cluster for testing and pulls the <i>test-use</i> image from this registry.	<code>docker.io</code>

PARAMETER	DESCRIPTION	DEFAULT
<code>textToSpeech.verification.image.repository</code>	The docker image repository that <code>helm test</code> uses to test <b>speech-to-text</b> service. Helm test pod uses this repository to pull <i>test-use</i> image.	<code>antsu/on-prem-client</code>
<code>textToSpeech.verification.image.tag</code>	The docker image tag used with <code>helm test</code> for <b>speech-to-text</b> service. Helm test pod uses this tag to pull <i>test-use</i> image.	<code>latest</code>
<code>textToSpeech.verification.image.pullByHash</code>	Whether the <i>test-use</i> docker image is pulled by hash. If <code>true</code> , <code>textToSpeech.verification.image.hash</code> should be added, with valid image hash value.	<code>false</code>
<code>textToSpeech.verification.image.arguments</code>	The arguments to execute with the <i>test-use</i> docker image. The helm test pod passes these arguments to container when running <code>helm test</code> .	<code>"/text-to-speech-client"</code> <code>--input='What's the weather like'</code> <code>--host=\$(TEXT_TO_SPEECH_HOST)</code> <code>--port=\$(TEXT_TO_SPEECH_PORT)"</code>

### Speech-to-Text (sub-chart: charts/speechToText)

To override the "umbrella" chart, add the prefix `speechToText.` on any parameter to make it more specific. For example, it will override the corresponding parameter for example, `speechToText.numberOfConcurrentRequest` overrides `numberOfConcurrentRequest`.

PARAMETER	DESCRIPTION	DEFAULT
<code>enabled</code>	Whether the <b>speech-to-text</b> service is enabled.	<code>false</code>
<code>numberOfConcurrentRequest</code>	The number of concurrent requests for the <b>speech-to-text</b> service. This chart automatically calculates CPU and memory resources, based on this value.	<code>2</code>
<code>optimizeForAudioFile</code>	Whether the service needs to optimize for audio input via audio files. If <code>true</code> , this chart will allocate more CPU resource to service.	<code>false</code>
<code>image.registry</code>	The <b>speech-to-text</b> docker image registry.	<code>containerpreview.azurecr.io</code>
<code>image.repository</code>	The <b>speech-to-text</b> docker image repository.	<code>microsoft/cognitive-services-speech-to-text</code>
<code>image.tag</code>	The <b>speech-to-text</b> docker image tag.	<code>latest</code>
<code>image.pullSecrets</code>	The image secrets for pulling the <b>speech-to-text</b> docker image.	

PARAMETER	DESCRIPTION	DEFAULT
<code>image.pullByHash</code>	Whether the docker image is pulled by hash. If <code>true</code> , <code>image.hash</code> is required.	<code>false</code>
<code>image.hash</code>	The <b>speech-to-text</b> docker image hash. Only used when <code>image.pullByHash: true</code> .	
<code>image.args.eula</code> (required)	Indicates you've accepted the license. The only valid value is <code>accept</code>	
<code>image.args.billing</code> (required)	The billing endpoint URI value is available on the Azure portal's Speech Overview page.	
<code>image.args.apikey</code> (required)	Used to track billing information.	
<code>service.type</code>	The Kubernetes service type of the <b>speech-to-text</b> service. See the <a href="#">Kubernetes service types instructions</a> for more details and verify cloud provider support.	<code>LoadBalancer</code>
<code>service.port</code>	The port of the <b>speech-to-text</b> service.	<code>80</code>
<code>service.annotations</code>	The <b>speech-to-text</b> annotations for the service metadata. Annotations are key value pairs. <code>annotations:</code> <code>some/annotation1: value1</code> <code>some/annotation2: value2</code>	
<code>service.autoScaler.enabled</code>	Whether the <a href="#">Horizontal Pod Autoscaler</a> is enabled. If <code>true</code> , the <code>speech-to-text-autoscaler</code> will be deployed in the Kubernetes cluster.	<code>true</code>
<code>service.podDisruption.enabled</code>	Whether the <a href="#">Pod Disruption Budget</a> is enabled. If <code>true</code> , the <code>speech-to-text-poddisruptionbudget</code> will be deployed in the Kubernetes cluster.	<code>true</code>

### Text-to-Speech (sub-chart: charts/textToSpeech)

To override the "umbrella" chart, add the prefix `textToSpeech.` on any parameter to make it more specific. For example, it will override the corresponding parameter for example, `textToSpeech.numberOfConcurrentRequest` overrides `numberOfConcurrentRequest`.

PARAMETER	DESCRIPTION	DEFAULT
<code>enabled</code>	Whether the <b>text-to-speech</b> service is enabled.	<code>false</code>

PARAMETER	DESCRIPTION	DEFAULT
<code>numberOfConcurrentRequest</code>	The number of concurrent requests for the <b>text-to-speech</b> service. This chart automatically calculates CPU and memory resources, based on this value.	2
<code>optimizeForTurboMode</code>	Whether the service needs to optimize for text input via text files. If <code>true</code> , this chart will allocate more CPU resource to service.	false
<code>image.registry</code>	The <b>text-to-speech</b> docker image registry.	containerpreview.azurecr.io
<code>image.repository</code>	The <b>text-to-speech</b> docker image repository.	microsoft/cognitive-services-text-to-speech
<code>image.tag</code>	The <b>text-to-speech</b> docker image tag.	latest
<code>image.pullSecrets</code>	The image secrets for pulling the <b>text-to-speech</b> docker image.	
<code>image.pullByHash</code>	Whether the docker image is pulled by hash. If <code>true</code> , <code>image.hash</code> is required.	false
<code>image.hash</code>	The <b>text-to-speech</b> docker image hash. Only used when <code>image.pullByHash: true</code> .	
<code>image.args.eula</code> (required)	Indicates you've accepted the license. The only valid value is <code>accept</code>	
<code>image.args.billing</code> (required)	The billing endpoint URI value is available on the Azure portal's Speech Overview page.	
<code>image.args.apikey</code> (required)	Used to track billing information.	
<code>service.type</code>	The Kubernetes service type of the <b>text-to-speech</b> service. See the <a href="#">Kubernetes service types instructions</a> for more details and verify cloud provider support.	LoadBalancer
<code>service.port</code>	The port of the <b>text-to-speech</b> service.	80
<code>service.annotations</code>	The <b>text-to-speech</b> annotations for the service metadata. Annotations are key value pairs. <code>annotations:</code> <code>some/annotation1: value1</code> <code>some/annotation2: value2</code>	

PARAMETER	DESCRIPTION	DEFAULT
<code>service.autoScaler.enabled</code>	Whether the <a href="#">Horizontal Pod Autoscaler</a> is enabled. If <code>true</code> , the <code>text-to-speech-autoscaler</code> will be deployed in the Kubernetes cluster.	<code>true</code>
<code>service.podDisruption.enabled</code>	Whether the <a href="#">Pod Disruption Budget</a> is enabled. If <code>true</code> , the <code>text-to-speech-poddisruptionbudget</code> will be deployed in the Kubernetes cluster.	<code>true</code>

## Next steps

For more details on installing applications with Helm in Azure Kubernetes Service (AKS), [visit here](#).

[Cognitive Services Containers](#)

# Deploy the Speech service container to Azure Container Instances

12/4/2019 • 4 minutes to read • [Edit Online](#)

Learn how to deploy the Cognitive Services [Speech service](#) container to Azure [Container Instances](#). This procedure demonstrates the creation of an Azure Speech service resource. Then we discuss pulling the associated container image. Finally, we highlight the ability to exercise the orchestration of the two from a browser. Using containers can shift the developers' attention away from managing infrastructure to instead focusing on application development.

## Prerequisites

- Use an Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- Install the [Azure CLI](#) (az).
- [Docker engine](#) and validate that the Docker CLI works in a console window.

## Request access to the container registry

You must first complete and submit the [Cognitive Services Speech Containers Request form](#) to request access to the container.

The form requests information about you, your company, and the user scenario for which you'll use the container. After you've submitted the form, the Azure Cognitive Services team reviews it to ensure that you meet the criteria for access to the private container registry.

### IMPORTANT

You must use an email address that's associated with either a Microsoft Account (MSA) or Azure Active Directory (Azure AD) account in the form.

If your request is approved, you'll receive an email with instructions that describe how to obtain your credentials and access the private container registry.

## Create a Speech resource

1. Sign into the [Azure portal](#)
2. Click [Create Speech](#) resource
3. Enter all required settings:

SETTING	VALUE
Name	Desired name (2-64 characters)
Subscription	Select appropriate subscription
Location	Select any nearby and available location
Pricing Tier	F0 - the minimal pricing tier
Resource Group	Select an available resource group

4. Click **Create** and wait for the resource to be created. After it is created, navigate to the resource page
5. Collect configured `endpoint` and an API key:

RESOURCE TAB IN PORTAL	SETTING	VALUE
Overview	Endpoint	Copy the endpoint. It looks similar to <code>https://speech.cognitiveservices.azure.com/sts/v1.</code>

RESOURCE TAB IN PORTAL	SETTING	VALUE
Keys	API Key	Copy 1 of the two keys. It is a 32 alphanumeric-character string with no spaces or dashes, xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx .

## Create an Azure Container Instance resource from the Azure CLI

The YAML below defines the Azure Container Instance resource. Copy and paste the contents into a new file, named `my-aci.yaml` and replace the commented values with your own. Refer to the [template format](#) for valid YAML. Refer to the [container repositories and images](#) for the available image names and their corresponding repository. For more information of the YAML reference for Container instances, see [YAML reference: Azure Container Instances](#).

```

apiVersion: 2018-10-01
location: # < Valid location >
name: # < Container Group name >
imageRegistryCredentials: # This is required when pulling a non-public image
 - server: containerpreview.azurecr.io
 username: # < The username for the preview container registry >
 password: # < The password for the preview container registry >
properties:
 containers:
 - name: # < Container name >
 properties:
 image: # < Repository/Image name >
 environmentVariables: # These env vars are required
 - name: eula
 value: accept
 - name: billing
 value: # < Service specific Endpoint URL >
 - name: apikey
 value: # < Service specific API key >
 resources:
 requests:
 cpu: 4 # Always refer to recommended minimal resources
 memoryInGb: 8 # Always refer to recommended minimal resources
 ports:
 - port: 5000
osType: Linux
volumes: # This node, is only required for container instances that pull their model in at runtime, such as LUIS.
 - name: aci-file-share
 azureFile:
 shareName: # < File share name >
 storageAccountName: # < Storage account name>
 storageAccountKey: # < Storage account key >
restartPolicy: OnFailure
ipAddress:
 type: Public
 ports:
 - protocol: tcp
 port: 5000
tags: null
type: Microsoft.ContainerInstance/containerGroups

```

### NOTE

Not all locations have the same CPU and Memory availability. Refer to the [location and resources](#) table for the listing of available resources for containers per location and OS.

We'll rely on the YAML file we created for the `az container create` command. From the Azure CLI, execute the `az container create` command replacing the `<resource-group>` with your own. Additionally, for securing values within a YAML deployment refer to [secure values](#).

```
az container create -g <resource-group> -f my-aci.yaml
```

The output of the command is `Running...` if valid, after sometime the output changes to a JSON string representing the newly created ACI resource. The container image is more than likely not be available for a while, but the resource is now deployed.

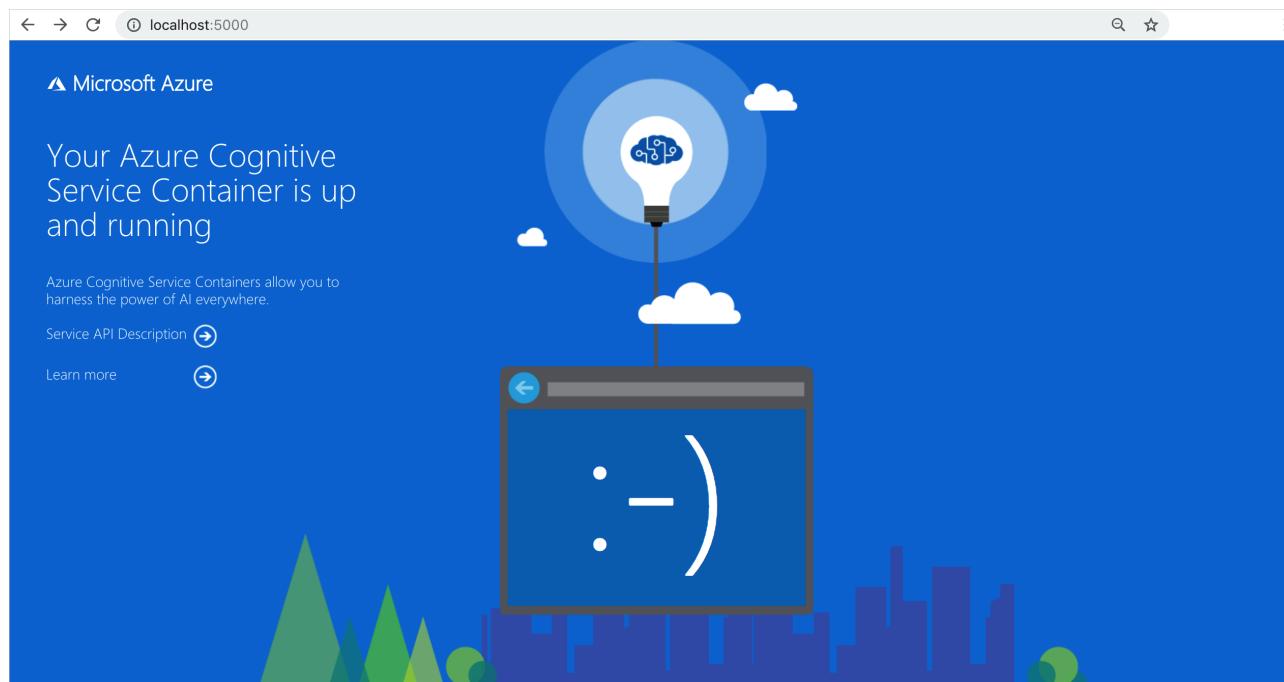
**TIP**

Pay close attention to the locations of public preview Azure Cognitive Service offerings, as the YAML will needed to be adjusted accordingly to match the location.

## Validate that a container is running

There are several ways to validate that the container is running. Locate the *External IP* address and exposed port of the container in question, and open your favorite web browser. Use the various request URLs below to validate the container is running. The example request URLs listed below are `http://localhost:5000`, but your specific container may vary. Keep in mind that you're to rely on your container's *External IP* address and exposed port.

REQUEST URL	PURPOSE
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/status</code>	Requested with an HTTP GET, to validate that the container is running without causing an endpoint query. This request can be used for Kubernetes <a href="#">liveness and readiness probes</a> .
<code>http://localhost:5000/swagger</code>	The container provides a full set of documentation for the endpoints and a <b>Try it out</b> feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.



## Next steps

Let's continue working with Azure Cognitive Services containers.

[Use more Cognitive Services Containers](#)

# Speech-to-text REST API

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

As an alternative to the [Speech SDK](#), the Speech service allows you to convert speech-to-text using a REST API. Each accessible endpoint is associated with a region. Your application requires a subscription key for the endpoint you plan to use.

Before using the speech-to-text REST API, understand:

- Requests that use the REST API and transmit audio directly can only contain up to 60 seconds of audio.
- The speech-to-text REST API only returns final results. Partial results are not provided.

If sending longer audio is a requirement for your application, consider using the [Speech SDK](#) or a file-based REST API, like [batch transcription](#).

## Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each service:

SUPPORTED AUTHORIZATION HEADERS	SPEECH-TO-TEXT	TEXT-TO-SPEECH
Ocp-Apim-Subscription-Key	Yes	No
Authorization: Bearer	Yes	Yes

When using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your subscription key. For example:

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your subscription key for an access token that's valid for 10 minutes. In the next few sections you'll learn how to get a token, and use a token.

### How to get an access token

To get an access token, you'll need to make a request to the `issueToken` endpoint using the `Ocp-Apim-Subscription-Key` and your subscription key.

These regions and endpoints are supported:

REGION	TOKEN SERVICE ENDPOINT
Australia East	<a href="https://australiaeast.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://australiaeast.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Canada Central	<a href="https://canadacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://canadacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Central US	<a href="https://centralus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://centralus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
East Asia	<a href="https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
East US	<a href="https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
East US 2	<a href="https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
France Central	<a href="https://francecentral.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://francecentral.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
India Central	<a href="https://centralindia.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://centralindia.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Japan East	<a href="https://japaneast.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://japaneast.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Korea Central	<a href="https://koreacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://koreacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
North Central US	<a href="https://northcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://northcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
North Europe	<a href="https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
South Central US	<a href="https://southcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://southcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Southeast Asia	<a href="https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
UK South	<a href="https://uksouth.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://uksouth.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>

REGION	TOKEN SERVICE ENDPOINT
West Europe	<a href="https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
West US	<a href="https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
West US 2	<a href="https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>

Use these samples to create your access token request.

#### HTTP sample

This example is a simple HTTP request to get a token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.api.cognitive.microsoft.com
Content-type: application/x-www-form-urlencoded
Content-Length: 0
```

The body of the response contains the access token in JSON Web Token (JWT) format.

#### PowerShell sample

This example is a simple PowerShell script to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
$FetchTokenHeader = @{
 'Content-type'='application/x-www-form-urlencoded';
 'Content-Length' = '0';
 'Ocp-Apim-Subscription-Key' = 'YOUR_SUBSCRIPTION_KEY'
}

$OAuthToken = Invoke-RestMethod -Method POST -Uri https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
-Headers $FetchTokenHeader

show the token received
$OAuthToken
```

#### cURL sample

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). This cURL command illustrates how to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
curl -v -X POST
"https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
-H "Content-type: application/x-www-form-urlencoded" \
-H "Content-Length: 0" \
-H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

#### C# sample

This C# class illustrates how to get an access token. Pass your Speech Service subscription key when you instantiate the class. If your subscription isn't in the West US region, change the value of `FetchTokenUri` to match the region for your subscription.

```

public class Authentication
{
 public static readonly string FetchTokenUri =
 "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
 private string subscriptionKey;
 private string token;

 public Authentication(string subscriptionKey)
 {
 this.subscriptionKey = subscriptionKey;
 this.token = FetchTokenAsync(FetchTokenUri, subscriptionKey).Result;
 }

 public string GetAccessToken()
 {
 return this.token;
 }

 private async Task<string> FetchTokenAsync(string fetchUri, string subscriptionKey)
 {
 using (var client = new HttpClient())
 {
 client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
 UriBuilder uriBuilder = new UriBuilder(fetchUri);

 var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
 Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
 return await result.Content.ReadAsStringAsync();
 }
 }
}

```

#### Python sample

```

Request module must be installed.
Run pip install requests if necessary.
import requests

subscription_key = 'REPLACE_WITH_YOUR_KEY'

def get_token(subscription_key):
 fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
 headers = {
 'Ocp-Apim-Subscription-Key': subscription_key
 }
 response = requests.post(fetch_token_url, headers=headers)
 access_token = str(response.text)
 print(access_token)

```

#### How to use an access token

The access token should be sent to the service as the `Authorization: Bearer <TOKEN>` header. Each access token is valid for 10 minutes. You can get a new token at any time, however, to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the text-to-speech REST API:

```

POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.stt.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

// Message body here...

```

## Regions and endpoints

These regions are supported for speech-to-text transcription using the REST API. Make sure that you select the endpoint that matches your subscription region.

REGION	ENDPOINT
Australia East	<a href="https://australiaeast.stt.speech.microsoft.com/speech/recognition/conversation/cc">https://australiaeast.stt.speech.microsoft.com/speech/recognition/conversation/cc</a>
Canada Central	<a href="https://canadacentral.stt.speech.microsoft.com/speech/recognition/conversation/cc">https://canadacentral.stt.speech.microsoft.com/speech/recognition/conversation/cc</a>
Central US	<a href="https://centralus.stt.speech.microsoft.com/speech/recognition/conversation/cognit">https://centralus.stt.speech.microsoft.com/speech/recognition/conversation/cognit</a>
East Asia	<a href="https://eastasia.stt.speech.microsoft.com/speech/recognition/conversation/cogniti">https://eastasia.stt.speech.microsoft.com/speech/recognition/conversation/cogniti</a>

REGION	ENDPOINT
East US	<a href="https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
East US 2	<a href="https://eastus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://eastus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
France Central	<a href="https://francecentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://francecentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
India Central	<a href="https://centralindia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://centralindia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Japan East	<a href="https://japaneast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://japaneast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Korea Central	<a href="https://koreacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://koreacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
North Central US	<a href="https://northcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://northcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
North Europe	<a href="https://northeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://northeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
South Central US	<a href="https://southcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://southcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Southeast Asia	<a href="https://southeastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://southeastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
UK South	<a href="https://uksouth.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://uksouth.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
West Europe	<a href="https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
West US	<a href="https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
West US 2	<a href="https://westus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://westus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>

#### NOTE

The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is: <https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US>.

## Query parameters

These parameters may be included in the query string of the REST request.

PARAMETER	DESCRIPTION	REQUIRED / OPTIONAL
<code>language</code>	Identifies the spoken language that is being recognized. See <a href="#">Supported languages</a> .	Required
<code>format</code>	Specifies the result format. Accepted values are <code>simple</code> and <code>detailed</code> . Simple results include <code>RecognitionStatus</code> , <code>DisplayText</code> , <code>Offset</code> , and <code>Duration</code> . Detailed responses include multiple results with confidence values and four different representations. The default setting is <code>simple</code> .	Optional
<code>profanity</code>	Specifies how to handle profanity in recognition results. Accepted values are <code>masked</code> , which replaces profanity with asterisks, <code>removed</code> , which removes all profanity from the result, or <code>raw</code> , which includes the profanity in the result. The default setting is <code>masked</code> .	Optional

## Request headers

This table lists required and optional headers for speech-to-text requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
<code>Ocp-Apim-Subscription-Key</code>	Your Speech service subscription key.	Either this header or <code>Authorization</code> is required.
<code>Authorization</code>	An authorization token preceded by the word <code>Bearer</code> . For more information, see <a href="#">Authentication</a> .	Either this header or <code>Ocp-Apim-Subscription-Key</code> is required.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
<code>Content-type</code>	Describes the format and codec of the provided audio data. Accepted values are <code>audio/wav; codecs=audio/pcm; samplerate=16000</code> and <code>audio/ogg; codecs=opus</code> .	Required
<code>Transfer-Encoding</code>	Specifies that chunked audio data is being sent, rather than a single file. Only use this header if chunking audio data.	Optional
<code>Expect</code>	If using chunked transfer, send <code>Expect: 100-continue</code> . The Speech service acknowledges the initial request and awaits additional data.	Required if sending chunked audio data.
<code>Accept</code>	If provided, it must be <code>application/json</code> . The Speech service provides results in JSON. Some request frameworks provide an incompatible default value. It is good practice to always include <code>Accept</code> .	Optional, but recommended.

## Audio formats

Audio is sent in the body of the HTTP `POST` request. It must be in one of the formats in this table:

FORMAT	CODEC	BITRATE	SAMPLE RATE
WAV	PCM	16-bit	16 kHz, mono
OGG	OPUS	16-bit	16 kHz, mono

### NOTE

The above formats are supported through REST API and WebSocket in the Speech service. The [Speech SDK](#) currently only supports the WAV format with PCM codec.

## Sample request

The sample below includes the hostname and required headers. It's important to note that the service also expects audio data, which is not included in this sample. As mentioned earlier, chunking is recommended, however, not required.

```
POST speech/recognition/conversation/cognitiveservices/v1?language=en-US&format=detailed HTTP/1.1
Accept: application/json;text/xml
Content-Type: audio/wav; codecs=audio/pcm; samplerate=16000
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.stt.speech.microsoft.com
Transfer-Encoding: chunked
Expect: 100-continue
```

## HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
100	Continue	The initial request has been accepted. Proceed with sending the rest of the data. (Used with chunked transfer.)
200	OK	The request was successful; the response body is a JSON object.
400	Bad request	Language code not provided, not a supported language, invalid audio file, etc.
401	Unauthorized	Subscription key or authorization token is invalid in the specified region, or invalid endpoint.
403	Forbidden	Missing subscription key or authorization token.

## Chunked transfer

Chunked transfer (`Transfer-Encoding: chunked`) can help reduce recognition latency. It allows the Speech service to begin processing the audio file while it is transmitted. The REST API does not provide partial or interim results.

This code sample shows how to send audio in chunks. Only the first chunk should contain the audio file's header. `request` is an `HttpWebRequest` object connected to the appropriate REST endpoint. `audioFile` is the path to an audio file on disk.

```
HttpWebRequest request = null;
request = (HttpWebRequest)HttpWebRequest.Create(requestUri);
request.SendChunked = true;
request.Accept = @"application/json;text/xml";
request.Method = "POST";
request.ProtocolVersion = HttpVersion.Version11;
request.Host = host;
request.ContentType = @"audio/wav; codecs=audio/pcm; samplerate=16000";
request.Headers["Ocp-Apim-Subscription-Key"] = args[1];
request.AllowWriteStreamBuffering = false;

using (fs = new FileStream(audioFile, FileMode.Open, FileAccess.Read))
{
 /*
 * Open a request stream and write 1024 byte chunks in the stream one at a time.
 */
 byte[] buffer = null;
 int bytesRead = 0;
 using (Stream requestStream = request.GetRequestStream())
 {
 /*
 * Read 1024 raw bytes from the input audio file.
 */
 buffer = new Byte[checked((uint)Math.Min(1024, (int)fs.Length))];
 while ((bytesRead = fs.Read(buffer, 0, buffer.Length)) != 0)
 {
 requestStream.Write(buffer, 0, bytesRead);
 }

 // Flush
 requestStream.Flush();
 }
}
```

## Response parameters

Results are provided as JSON. The `simple` format includes these top-level fields.

PARAMETER	DESCRIPTION
<code>RecognitionStatus</code>	Status, such as <code>Success</code> for successful recognition. See next table.
<code>DisplayText</code>	The recognized text after capitalization, punctuation, inverse text normalization (conversion of spoken text to shorter forms, such as 200 for "two hundred" or "Dr. Smith" for "doctor smith"), and profanity masking. Present only on success.
<code>Offset</code>	The time (in 100-nanosecond units) at which the recognized speech begins in the audio stream.
<code>Duration</code>	The duration (in 100-nanosecond units) of the recognized speech in the audio stream.

The `RecognitionStatus` field may contain these values:

STATUS	DESCRIPTION
<code>Success</code>	The recognition was successful and the <code>DisplayText</code> field is present.
<code>NoMatch</code>	Speech was detected in the audio stream, but no words from the target language were matched. Usually means the recognition language is a different language from the one the user is speaking.
<code>InitialSilenceTimeout</code>	The start of the audio stream contained only silence, and the service timed out waiting for speech.
<code>BabbleTimeout</code>	The start of the audio stream contained only noise, and the service timed out waiting for speech.
<code>Error</code>	The recognition service encountered an internal error and could not continue. Try again if possible.

#### NOTE

If the audio consists only of profanity, and the `profanity` query parameter is set to `remove`, the service does not return a speech result.

The `detailed` format includes the same data as the `simple` format, along with `NBest`, a list of alternative interpretations of the same recognition result. These results are ranked from most likely to least likely. The first entry is the same as the main recognition result. When using the `detailed` format, `DisplayText` is provided as `Display` for each result in the `NBest` list.

Each object in the `NBest` list includes:

PARAMETER	DESCRIPTION
<code>Confidence</code>	The confidence score of the entry from 0.0 (no confidence) to 1.0 (full confidence)
<code>Lexical</code>	The lexical form of the recognized text: the actual words recognized.
<code>ITN</code>	The inverse-text-normalized ("canonical") form of the recognized text, with phone numbers, numbers, abbreviations ("doctor smith" to "dr smith"), and other transformations applied.
<code>MaskedITN</code>	The ITN form with profanity masking applied, if requested.
<code>Display</code>	The display form of the recognized text, with punctuation and capitalization added. This parameter is the same as <code>DisplayText</code> provided when format is set to <code>simple</code> .

## Sample responses

A typical response for `simple` recognition:

```
{
 "RecognitionStatus": "Success",
 "DisplayText": "Remind me to buy 5 pencils.",
 "Offset": "1236645672289",
 "Duration": "1236645672289"
}
```

A typical response for `detailed` recognition:

```
{
 "RecognitionStatus": "Success",
 "Offset": "1236645672289",
 "Duration": "1236645672289",
 "NBest": [
 {
 "Confidence" : "0.87",
 "Lexical" : "remind me to buy five pencils",
 "ITN" : "remind me to buy 5 pencils",
 "MaskedITN" : "remind me to buy 5 pencils",
 "Display" : "Remind me to buy 5 pencils.",
 },
 {
 "Confidence" : "0.54",
 "Lexical" : "rewind me to buy five pencils",
 "ITN" : "rewind me to buy 5 pencils",
 "MaskedITN" : "rewind me to buy 5 pencils",
 "Display" : "Rewind me to buy 5 pencils.",
 }
]
}
```

## Next steps

- [Get your Speech trial subscription](#)
- [Customize acoustic models](#)
- [Customize language models](#)

# Text-to-speech REST API

12/10/2019 • 9 minutes to read • [Edit Online](#)

The Speech service allows you to [convert text into synthesized speech](#) and [get a list of supported voices](#) for a region using a set of REST APIs. Each available endpoint is associated with a region. A subscription key for the endpoint/region you plan to use is required.

The text-to-speech REST API supports neural and standard text-to-speech voices, each of which supports a specific language and dialect, identified by locale.

- For a complete list of voices, see [language support](#).
- For information about regional availability, see [regions](#).

## IMPORTANT

Costs vary for standard, custom, and neural voices. For more information, see [Pricing](#).

Before using this API, understand:

- The text-to-speech REST API requires an Authorization header. This means that you need to complete a token exchange to access the service. For more information, see [Authentication](#).

## Authentication

Each request requires an authorization header. This table illustrates which headers are supported for each service:

SUPPORTED AUTHORIZATION HEADERS	SPEECH-TO-TEXT	TEXT-TO-SPEECH
Ocp-Apim-Subscription-Key	Yes	No
Authorization: Bearer	Yes	Yes

When using the `Ocp-Apim-Subscription-Key` header, you're only required to provide your subscription key. For example:

```
'Ocp-Apim-Subscription-Key': 'YOUR_SUBSCRIPTION_KEY'
```

When using the `Authorization: Bearer` header, you're required to make a request to the `issueToken` endpoint. In this request, you exchange your subscription key for an access token that's valid for 10 minutes. In the next few sections you'll learn how to get a token, and use a token.

### How to get an access token

To get an access token, you'll need to make a request to the `issueToken` endpoint using the `Ocp-Apim-Subscription-Key` and your subscription key.

These regions and endpoints are supported:

REGION	TOKEN SERVICE ENDPOINT
Australia East	<a href="https://australiaeast.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://australiaeast.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Canada Central	<a href="https://canadacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://canadacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Central US	<a href="https://centralus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://centralus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
East Asia	<a href="https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://eastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
East US	<a href="https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://eastus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>

REGION	TOKEN SERVICE ENDPOINT
East US 2	<a href="https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://eastus2.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
France Central	<a href="https://francecentral.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://francecentral.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
India Central	<a href="https://centralindia.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://centralindia.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Japan East	<a href="https://japaneast.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://japaneast.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Korea Central	<a href="https://koreacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://koreacentral.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
North Central US	<a href="https://northcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://northcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
North Europe	<a href="https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://northeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
South Central US	<a href="https://southcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://southcentralus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
Southeast Asia	<a href="https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://southeastasia.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
UK South	<a href="https://uksouth.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://uksouth.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
West Europe	<a href="https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://westeurope.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
West US	<a href="https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>
West US 2	<a href="https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken">https://westus2.api.cognitive.microsoft.com/sts/v1.0/issueToken</a>

Use these samples to create your access token request.

#### HTTP sample

This example is a simple HTTP request to get a token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. If your subscription isn't in the West US region, replace the `Host` header with your region's host name.

```
POST /sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY
Host: westus.api.cognitive.microsoft.com
Content-type: application/x-www-form-urlencoded
Content-Length: 0
```

The body of the response contains the access token in JSON Web Token (JWT) format.

#### PowerShell sample

This example is a simple PowerShell script to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct endpoint for the region that matches your subscription. This example is currently set to West US.

```
$FetchTokenHeader = @{
 'Content-type'='application/x-www-form-urlencoded';
 'Content-Length'='0';
 'Ocp-Apim-Subscription-Key' = 'YOUR_SUBSCRIPTION_KEY'
}

$OAuthToken = Invoke-RestMethod -Method POST -Uri https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken
-Headers $FetchTokenHeader

show the token received
$OAuthToken
```

#### cURL sample

cURL is a command-line tool available in Linux (and in the Windows Subsystem for Linux). This cURL command illustrates how to get an access token. Replace `YOUR_SUBSCRIPTION_KEY` with your Speech Service subscription key. Make sure to use the correct

endpoint for the region that matches your subscription. This example is currently set to West US.

```
curl -v -X POST
"https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
-H "Content-type: application/x-www-form-urlencoded" \
-H "Content-Length: 0" \
-H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

#### C# sample

This C# class illustrates how to get an access token. Pass your Speech Service subscription key when you instantiate the class. If your subscription isn't in the West US region, change the value of `FetchTokenUri` to match the region for your subscription.

```
public class Authentication
{
 public static readonly string FetchTokenUri =
 "https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken";
 private string subscriptionKey;
 private string token;

 public Authentication(string subscriptionKey)
 {
 this.subscriptionKey = subscriptionKey;
 this.token = FetchTokenAsync(FetchTokenUri, subscriptionKey).Result;
 }

 public string GetAccessToken()
 {
 return this.token;
 }

 private async Task<string> FetchTokenAsync(string fetchUri, string subscriptionKey)
 {
 using (var client = new HttpClient())
 {
 client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
 UriBuilder uriBuilder = new UriBuilder(fetchUri);

 var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
 Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
 return await result.Content.ReadAsStringAsync();
 }
 }
}
```

#### Python sample

```
Request module must be installed.
Run pip install requests if necessary.
import requests

subscription_key = 'REPLACE_WITH_YOUR_KEY'

def get_token(subscription_key):
 fetch_token_url = 'https://westus.api.cognitive.microsoft.com/sts/v1.0/issueToken'
 headers = {
 'Ocp-Apim-Subscription-Key': subscription_key
 }
 response = requests.post(fetch_token_url, headers=headers)
 access_token = str(response.text)
 print(access_token)
```

#### How to use an access token

The access token should be sent to the service as the `Authorization: Bearer <TOKEN>` header. Each access token is valid for 10 minutes. You can get a new token at any time, however, to minimize network traffic and latency, we recommend using the same token for nine minutes.

Here's a sample HTTP request to the text-to-speech REST API:

```

POST /cognitiveservices/v1 HTTP/1.1
Authorization: Bearer YOUR_ACCESS_TOKEN
Host: westus.stt.speech.microsoft.com
Content-type: application/ssml+xml
Content-Length: 199
Connection: Keep-Alive

// Message body here...

```

## Get a list of voices

The `voices/list` endpoint allows you to get a full list of voices for a specific region/endpoint.

### Regions and endpoints

REGION	ENDPOINT
Australia East	<a href="https://australiaeast.tts.speech.microsoft.com/cognitiveservices/voices/list">https://australiaeast.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
Brazil South	<a href="https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/voices/list">https://brazilsouth.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
Canada Central	<a href="https://canadacentral.tts.speech.microsoft.com/cognitiveservices/voices/list">https://canadacentral.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
Central US	<a href="https://centralus.tts.speech.microsoft.com/cognitiveservices/voices/list">https://centralus.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
East Asia	<a href="https://eastasia.tts.speech.microsoft.com/cognitiveservices/voices/list">https://eastasia.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
East US	<a href="https://eastus.tts.speech.microsoft.com/cognitiveservices/voices/list">https://eastus.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
East US 2	<a href="https://eastus2.tts.speech.microsoft.com/cognitiveservices/voices/list">https://eastus2.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
France Central	<a href="https://francecentral.tts.speech.microsoft.com/cognitiveservices/voices/list">https://francecentral.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
India Central	<a href="https://centralindia.tts.speech.microsoft.com/cognitiveservices/voices/list">https://centralindia.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
Japan East	<a href="https://japaneast.tts.speech.microsoft.com/cognitiveservices/voices/list">https://japaneast.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
Korea Central	<a href="https://koreacentral.tts.speech.microsoft.com/cognitiveservices/voices/list">https://koreacentral.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
North Central US	<a href="https://northcentralus.tts.speech.microsoft.com/cognitiveservices/voices/list">https://northcentralus.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
North Europe	<a href="https://northeurope.tts.speech.microsoft.com/cognitiveservices/voices/list">https://northeurope.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
South Central US	<a href="https://southcentralus.tts.speech.microsoft.com/cognitiveservices/voices/list">https://southcentralus.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
Southeast Asia	<a href="https://southeastasia.tts.speech.microsoft.com/cognitiveservices/voices/list">https://southeastasia.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
UK South	<a href="https://uksouth.tts.speech.microsoft.com/cognitiveservices/voices/list">https://uksouth.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
West Europe	<a href="https://westeurope.tts.speech.microsoft.com/cognitiveservices/voices/list">https://westeurope.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
West US	<a href="https://westus.tts.speech.microsoft.com/cognitiveservices/voices/list">https://westus.tts.speech.microsoft.com/cognitiveservices/voices/list</a>
West US 2	<a href="https://westus2.tts.speech.microsoft.com/cognitiveservices/voices/list">https://westus2.tts.speech.microsoft.com/cognitiveservices/voices/list</a>

### Request headers

This table lists required and optional headers for text-to-speech requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
Authorization	An authorization token preceded by the word <code>Bearer</code> . For more information, see <a href="#">Authentication</a> .	Required

## Request body

A body isn't required for `GET` requests to this endpoint.

## Sample request

This request only requires an authorization header.

```
GET /cognitiveservices/voices/list HTTP/1.1
Host: westus.tts.speech.microsoft.com
Authorization: Bearer [Base64 access_token]
```

## Sample response

This response has been truncated to illustrate the structure of a response.

### NOTE

Voice availability varies by region/endpoint.

```
[
 {
 "Name": "Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)",
 "ShortName": "ar-EG-Hoda",
 "Gender": "Female",
 "Locale": "ar-EG"
 },
 {
 "Name": "Microsoft Server Speech Text to Speech Voice (ar-SA, Naayf)",
 "ShortName": "ar-SA-Naayf",
 "Gender": "Male",
 "Locale": "ar-SA"
 },
 {
 "Name": "Microsoft Server Speech Text to Speech Voice (bg-BG, Ivan)",
 "ShortName": "bg-BG-Ivan",
 "Gender": "Male",
 "Locale": "bg-BG"
 },
 {
 "Name": "Microsoft Server Speech Text to Speech Voice (ca-ES, HerenaRUS)",
 "ShortName": "ca-ES-HerenaRUS",
 "Gender": "Female",
 "Locale": "ca-ES"
 },
 {
 "Name": "Microsoft Server Speech Text to Speech Voice (cs-CZ, Jakub)",
 "ShortName": "cs-CZ-Jakub",
 "Gender": "Male",
 "Locale": "cs-CZ"
 },
 ...
]
```

## HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
200	OK	The request was successful.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
429	Too Many Requests	You have exceeded the quota or rate of requests allowed for your subscription.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

## Convert text-to-speech

The `v1` endpoint allows you to convert text-to-speech using [Speech Synthesis Markup Language \(SSML\)](#).

### Regions and endpoints

These regions are supported for text-to-speech using the REST API. Make sure that you select the endpoint that matches your subscription region.

### Standard and neural voices

Use this table to determine availability of standard and neural voices by region/endpoint:

REGION	ENDPOINT	STANDARD VOICES	NEURAL VOICES
Australia East	<a href="https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1">https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
Canada Central	<a href="https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1">https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
Central US	<a href="https://centralus.tts.speech.microsoft.com/cognitiveservices/v1">https://centralus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
East Asia	<a href="https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1">https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
East US	<a href="https://eastus.tts.speech.microsoft.com/cognitiveservices/v1">https://eastus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
East US 2	<a href="https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1">https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
France Central	<a href="https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1">https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
India Central	<a href="https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1">https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
Japan East	<a href="https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1">https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
Korea Central	<a href="https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1">https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
North Central US	<a href="https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1">https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
North Europe	<a href="https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1">https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
South Central US	<a href="https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1">https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
Southeast Asia	<a href="https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1">https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
UK South	<a href="https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1">https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes

REGION	ENDPOINT	STANDARD VOICES	NEURAL VOICES
West Europe	<a href="https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1">https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	
West US	<a href="https://westus.tts.speech.microsoft.com/cognitiveservices/v1">https://westus.tts.speech.microsoft.com/cognitiveservices/v1</a>	No	
West US 2	<a href="https://westus2.tts.speech.microsoft.com/cognitiveservices/v1">https://westus2.tts.speech.microsoft.com/cognitiveservices/v1</a>		Yes

## Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	<a href="https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Canada Central	<a href="https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Central US	<a href="https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
East Asia	<a href="https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
East US	<a href="https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
East US 2	<a href="https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
France Central	<a href="https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
India Central	<a href="https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Japan East	<a href="https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Korea Central	<a href="https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
North Central US	<a href="https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
North Europe	<a href="https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
South Central US	<a href="https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Southeast Asia	<a href="https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
UK South	<a href="https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
West Europe	<a href="https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
West US	<a href="https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>

REGION	ENDPOINT
West US 2	<a href="https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>

## Request headers

This table lists required and optional headers for text-to-speech requests.

HEADER	DESCRIPTION	REQUIRED / OPTIONAL
<code>Authorization</code>	An authorization token preceded by the word <code>Bearer</code> . For more information, see <a href="#">Authentication</a> .	Required
<code>Content-Type</code>	Specifies the content type for the provided text. Accepted value: <code>application/ssml+xml</code> .	Required
<code>X-Microsoft-OutputFormat</code>	Specifies the audio output format. For a complete list of accepted values, see <a href="#">audio outputs</a> .	Required
<code>User-Agent</code>	The application name. The value provided must be less than 255 characters.	Required

## Audio outputs

This is a list of supported audio formats that are sent in each request as the `X-Microsoft-OutputFormat` header. Each incorporates a bitrate and encoding type. The Speech service supports 24 kHz, 16 kHz, and 8 kHz audio outputs.

<code>raw-16khz-16bit-mono-pcm</code>	<code>raw-8khz-8bit-mono-mulaw</code>
<code>riff-8khz-8bit-mono-alaw</code>	<code>riff-8khz-8bit-mono-mulaw</code>
<code>riff-16khz-16bit-mono-pcm</code>	<code>audio-16khz-128kbitrate-mono-mp3</code>
<code>audio-16khz-64kbitrate-mono-mp3</code>	<code>audio-16khz-32kbitrate-mono-mp3</code>
<code>raw-24khz-16bit-mono-pcm</code>	<code>riff-24khz-16bit-mono-pcm</code>
<code>audio-24khz-160kbitrate-mono-mp3</code>	<code>audio-24khz-96kbitrate-mono-mp3</code>
<code>audio-24khz-48kbitrate-mono-mp3</code>	

### NOTE

If your selected voice and output format have different bit rates, the audio is resampled as necessary. However, 24 kHz voices do not support `audio-16khz-16kbps-mono-siren` and `riff-16khz-16kbps-mono-siren` output formats.

## Request body

The body of each `POST` request is sent as [Speech Synthesis Markup Language \(SSML\)](#). SSML allows you to choose the voice and language of the synthesized speech returned by the text-to-speech service. For a complete list of supported voices, see [language support](#).

### NOTE

If using a custom voice, the body of a request can be sent as plain text (ASCII or UTF-8).

## Sample request

This HTTP request uses SSML to specify the voice and language. The body cannot exceed 1,000 characters.

```
POST /cognitiveservices/v1 HTTP/1.1

X-Microsoft-OutputFormat: raw-16khz-16bit-mono-pcm
Content-Type: application/ssml+xml
Host: westus.tts.speech.microsoft.com
Content-Length: 225
Authorization: Bearer [Base64 access_token]

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Female'
name='en-US-JessaRUS'>
 Microsoft Speech Service Text-to-Speech API
</voice></speak>
```

See our quickstarts for language-specific examples:

- [.NET Core, C#](#)
- [Python](#)
- [Node.js](#)

#### HTTP status codes

The HTTP status code for each response indicates success or common errors.

HTTP STATUS CODE	DESCRIPTION	POSSIBLE REASON
200	OK	The request was successful; the response body is an audio file.
400	Bad Request	A required parameter is missing, empty, or null. Or, the value passed to either a required or optional parameter is invalid. A common issue is a header that is too long.
401	Unauthorized	The request is not authorized. Check to make sure your subscription key or token is valid and in the correct region.
413	Request Entity Too Large	The SSML input is longer than 1024 characters.
415	Unsupported Media Type	It's possible that the wrong Content-Type was provided. Content-Type should be set to application/ssml+xml .
429	Too Many Requests	You have exceeded the quota or rate of requests allowed for your subscription.
502	Bad Gateway	Network or server-side issue. May also indicate invalid headers.

If the HTTP status is 200 OK, the body of the response contains an audio file in the requested format. This file can be played as it's transferred, saved to a buffer, or saved to a file.

## Next steps

- [Get your Speech trial subscription](#)
- [Customize acoustic models](#)
- [Customize language models](#)

# Swagger documentation

12/4/2019 • 2 minutes to read • [Edit Online](#)

The Speech service offers a Swagger specification to interact with a handful of REST APIs used to import data, create models, test model accuracy, create custom endpoints, queue up batch transcriptions, and manage subscriptions. Most operations available through the Custom Speech portal can be completed programmatically using these APIs.

## NOTE

Both Speech-to-Text and Text-to-Speech operations are supported available as REST APIs, which are in turn documented in the Swagger specification.

## Generating code from the Swagger specification

The [Swagger specification](#) has options that allow you to quickly test for various paths. However, sometimes it's desirable to generate code for all paths, creating a single library of calls that you can base future solutions on. Let's take a look at the process to generate a Python library.

You'll need to set Swagger to the same region as your Speech service subscription. You can confirm your region in the Azure portal under your Speech service resource. For a complete list of supported regions, see [Regions](#).

1. Go to <https://editor.swagger.io>
2. Click **File**, then click **Import**
3. Enter the swagger URL including the region for your Speech service subscription  
`https://<your-region>.cris.ai/docs/v2.0/swagger`
4. Click **Generate Client** and select Python
5. Save the client library

You can use the Python library that you generated with the [Speech service samples on GitHub](#).

## Reference docs

- [REST \(Swagger\): Batch transcription and customization](#)
- [REST API: Speech-to-text](#)
- [REST API: Text-to-speech](#)

## Next steps

- [Speech service samples on GitHub](#).
- [Get a Speech service subscription key for free](#)

# About the Speech SDK

12/16/2019 • 3 minutes to read • [Edit Online](#)

The Speech Software Development Kit (SDK) gives your applications access to the functions of the Speech service, making it easier to develop speech-enabled software. Currently, the SDKs provide access to **speech-to-text**, **text-to-speech**, **speech translation**, **intent recognition**, and **Bot Framework's Direct Line Speech channel**.

You can easily capture audio from a microphone, read from a stream, or access audio files from storage with the Speech SDK. The Speech SDK supports WAV/PCM 16-bit, 16 kHz/8 kHz, single-channel audio for speech recognition. Additional audio formats are supported using the [speech-to-text REST endpoint](#) or the [batch transcription service](#).

A general overview about the capabilities and supported platforms can be found on the documentation [entry page](#).

PROGRAMMING LANGUAGE	PLATFORM	API REFERENCE
C/C++	Windows, Linux, macOS	<a href="#">Browse</a>
C#	Windows, UWP, .NET Framework (Windows), .NET Core, Unity	<a href="#">Browse</a>
Java	Android, Windows, Linux, macOS	<a href="#">Browse</a>
Java*	<a href="#">Speech Devices SDK</a>	<a href="#">Browse</a>
JavaScript/Node.js	Browser, Windows, Linux, macOS	<a href="#">Browse</a>
Objective-C	iOS, macOS	<a href="#">Browse</a>
Python	Windows, Linux, macOS	<a href="#">Browse</a>

\* The Java SDK is also available as part of the [Speech Devices SDK](#).

## IMPORTANT

By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

## Get the SDK

### Windows

For Windows, we support the following languages:

- C# (UWP and .NET), C++: You can reference and use the latest version of our Speech SDK NuGet package. The package includes 32-bit and 64-bit client libraries and managed (.NET) libraries. The SDK can be installed in Visual Studio by using NuGet, [Microsoft.CognitiveServices.Speech](#).

- Java: You can reference and use the latest version of our Speech SDK Maven package, which supports only Windows x64. In your Maven project, add

<https://csspeechstorage.blob.core.windows.net/maven/> as an additional repository and reference `com.microsoft.cognitiveservices.speech:client-sdk:1.8.0` as a dependency.

## Linux

### NOTE

Currently, we only support Ubuntu 16.04, Ubuntu 18.04, and Debian 9 on the following target architectures:

- x86, x64, and ARM64 for C++ development
- x64 and ARM64 for Java
- x64 for .NET Core and Python

Make sure you have the required libraries installed by running the following shell commands:

On Ubuntu:

```
sudo apt-get update
sudo apt-get install libssl1.0.0 libasound2
```

On Debian 9:

```
sudo apt-get update
sudo apt-get install libssl1.0.2 libasound2
```

- C#: You can reference and use the latest version of our Speech SDK NuGet package. To reference the SDK, add the following package reference to your project:

```
<PackageReference Include="Microsoft.CognitiveServices.Speech" Version="1.8.0" />
```

- Java: You can reference and use the latest version of our Speech SDK Maven package. In your Maven project, add <https://csspeechstorage.blob.core.windows.net/maven/> as an additional repository and reference

`com.microsoft.cognitiveservices.speech:client-sdk:1.7.0` as a dependency.

- C++: Download the SDK as a [tar package](#) and unpack the files in a directory of your choice. The following table shows the SDK folder structure:

PATH	DESCRIPTION
<code>license.md</code>	License
<code>ThirdPartyNotices.md</code>	Third-party notices
<code>include</code>	Header files for C and C++
<code>lib/x64</code>	Native x64 library for linking with your application

PATH	DESCRIPTION
lib/x86	Native x86 library for linking with your application

To create an application, copy or move the required binaries (and libraries) into your development environment. Include them as required in your build process.

## Android

The Java SDK for Android is packaged as an [AAR \(Android Library\)](#), which includes the necessary libraries and required Android permissions. It's hosted in a Maven repository at <https://csspeechstorage.blob.core.windows.net/maven/> as package `com.microsoft.cognitiveservices.speech:client-sdk:1.7.0`.

To consume the package from your Android Studio project, make the following changes:

- In the project-level build.gradle file, add the following to the `repository` section:

```
maven { url 'https://csspeechstorage.blob.core.windows.net/maven/' }
```

- In the module-level build.gradle file, add the following to the `dependencies` section:

```
implementation 'com.microsoft.cognitiveservices.speech:client-sdk:1.7.0'
```

The Java SDK is also part of the [Speech Devices SDK](#).

## Get the samples

For the latest samples, see the [Cognitive Services Speech SDK sample code repository](#) on GitHub.

## Next steps

- [Get your Speech trial subscription](#)
- [See how to recognize speech in C#](#)

# Scenario Availability

12/4/2019 • 2 minutes to read • [Edit Online](#)

The Speech SDK features many scenarios across a wide variety of programming languages and environments. Not all scenarios are available in all programming languages or all environments yet. Listed below is the availability of each scenario.

- **Speech-Recognition (SR), Phrase List, Intent, Translation, and On-premises containers**

- All programming languages/environments where there is an arrow link ↗ in the quickstart table [here](#).

- **Text-to-Speech (TTS)**

- C++/Windows & Linux
- C#/Windows & UWP & Unity
- Java (Jre and Android)
- Python
- Swift
- Objective-C
- TTS REST API can be used in every other situation.

- **Keyword Spotting (KWS)**

- C++/Windows & Linux
- C#/Windows & Linux
- Python/Windows & Linux
- Java/Windows & Linux & Android (Speech Devices SDK)
- Keyword spotting (KWS) functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK

- **Voice assistants**

- C++/Windows & Linux & macOS
- C#/Windows
- Java/Windows & Linux & macOS & Android (Speech Devices SDK)

- **Conversation Transcription**

- C++/Windows & Linux
- C# (Framework & .NET Core)/Windows & UWP & Linux
- Java/Windows & Linux & Android (Speech Devices SDK)

- **Call Center Transcription**

- REST API and can be used in any situation

- **Codec Compressed Audio Input**

- C++/Linux
- C#/Linux
- Java/Linux, Android, and iOS

# Quickstart: Create a project

11/20/2019 • 13 minutes to read • [Edit Online](#)

In this quickstart, you'll create an empty project for your preferred programming language that you'll use to complete a quickstart or create an application.

## Choose your target environment

- [Visual Studio](#)
- [Unity](#)
- [UWP](#)
- [Xamarin](#)

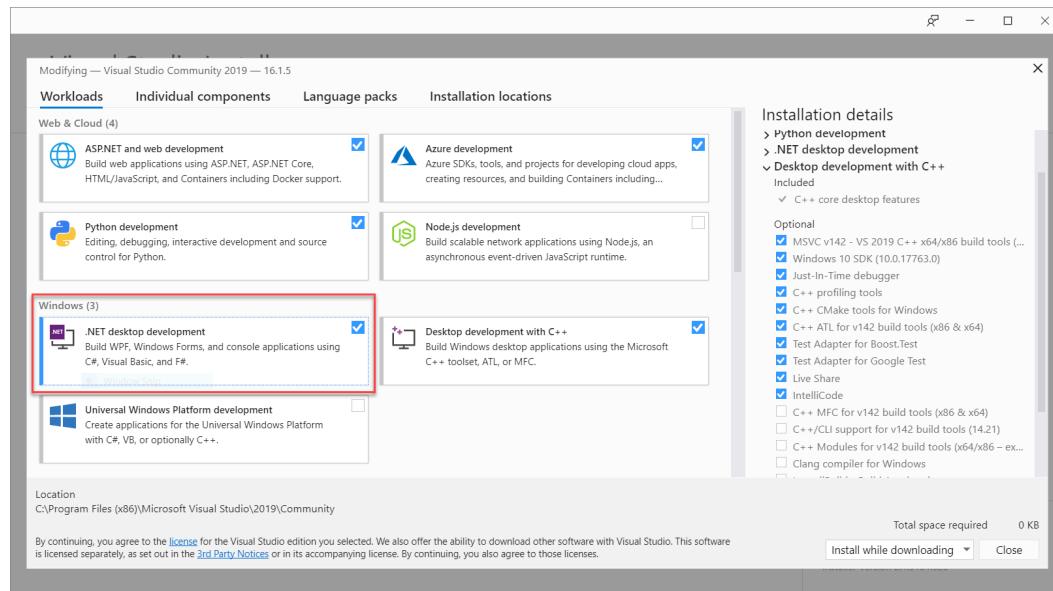
To create a Visual Studio project for Windows development, you need to create the project, set up Visual Studio for .NET desktop development, install the Speech SDK, and choose the target architecture.

## Create the project and add the workload

To start, create the project in Visual Studio, and make sure that Visual Studio is set up for .NET desktop development:

1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.
3. In the **Create a new project** window, choose **Console App (.NET Framework)**, and then select **Next**.
4. In the **Configure your new project** window, enter *helloworld* in **Project name**, choose or create the directory path in **Location**, and then select **Create**.
5. From the Visual Studio menu bar, select **Tools > Get Tools and Features**, which opens Visual Studio Installer and displays the **Modifying** dialog box.
6. Check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, select the check box next to it, and then select **Modify** to start the installation. It may take a few minutes to download and install.

If the check box next to **.NET desktop development** is already selected, select **Close** to exit the dialog box.

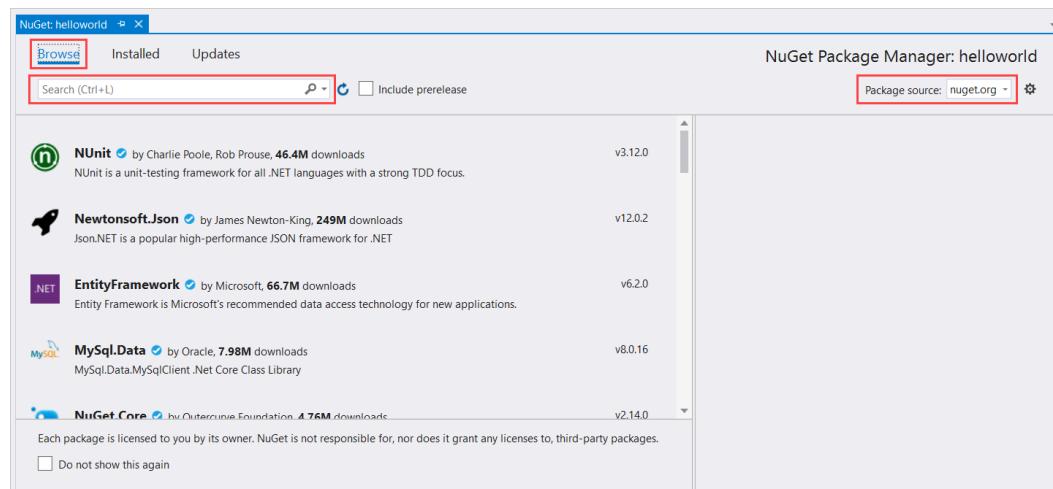


## 7. Close Visual Studio Installer.

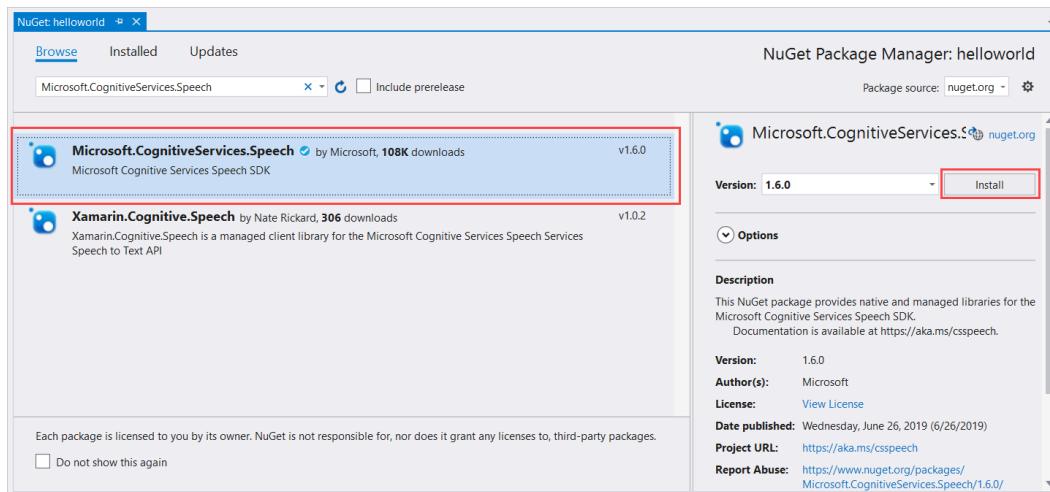
### Install the Speech SDK

The next step is to install the [Speech SDK NuGet package](#), so you can reference it in the code.

- In the Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show the NuGet Package Manager.



- In the upper-right corner, find the **Package Source** drop-down box, and make sure that **nuget.org** is selected.
- In the upper-left corner, select **Browse**.
- In the search box, type *Microsoft.CognitiveServices.Speech* and select **Enter**.
- From the search results, select the **Microsoft.CognitiveServices.Speech** package, and then select **Install** to install the latest stable version.



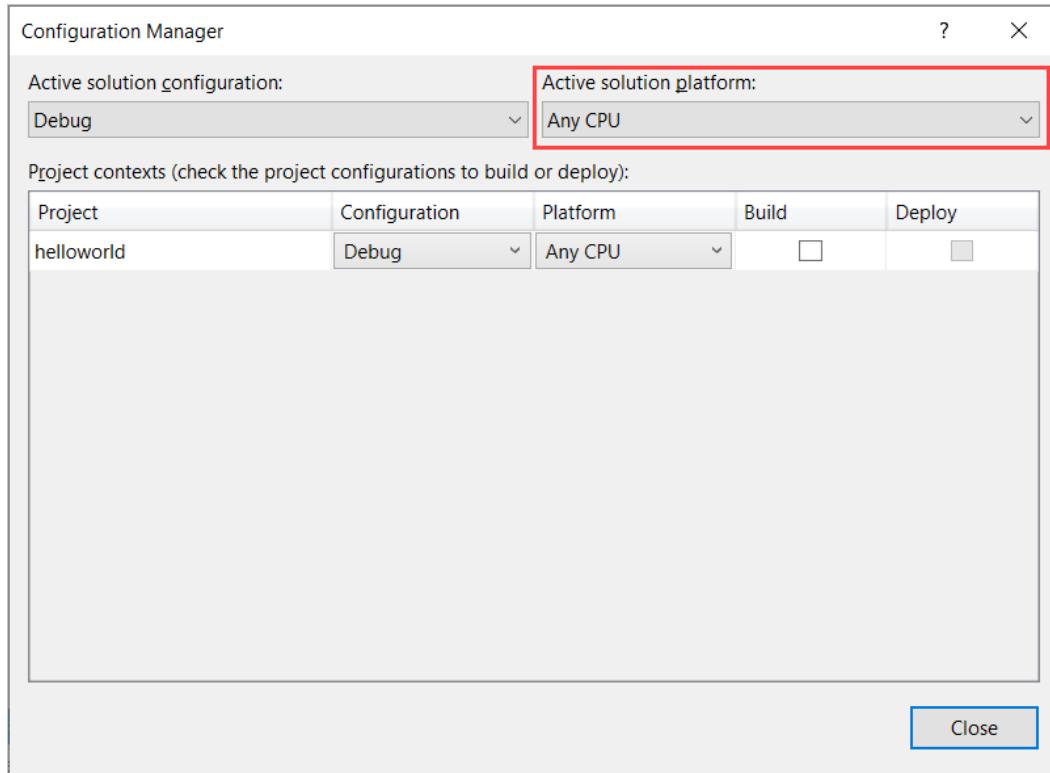
6. Accept all agreements and licenses to start the installation.

After the package is installed, a confirmation appears in the **Package Manager Console** window.

### Choose the target architecture

Now, to build and run the console application, create a platform configuration matching your computer's architecture.

1. From the menu bar, select **Build > Configuration Manager**. The **Configuration Manager** dialog box appears.



2. In the **Active solution platform** drop-down box, select **New**. The **New Solution Platform** dialog box appears.
3. In the **Type or select the new platform** drop-down box:
  - If you're running 64-bit Windows, select **x64**.
  - If you're running 32-bit Windows, select **x86**.
4. Select **OK** and then **Close**.

## Choose your target environment

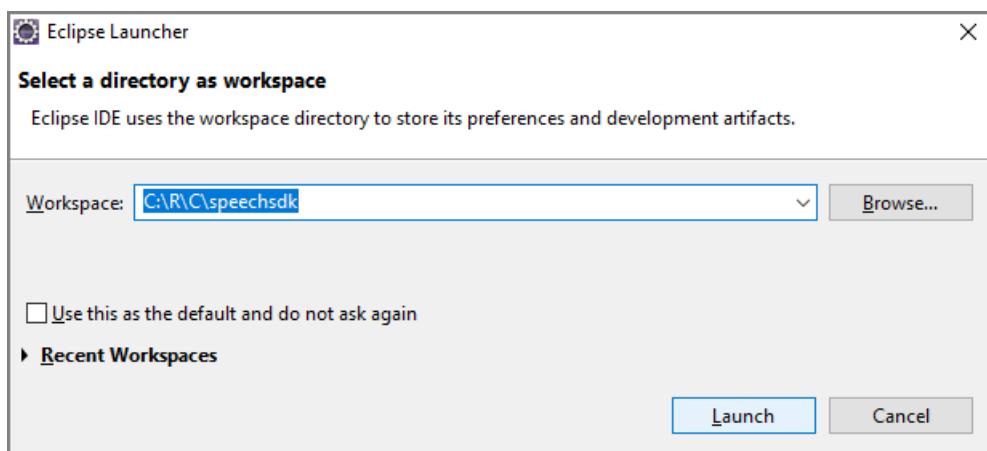
- Linux
- macOS
- Windows

For this sample, we'll be compiling with g++, so all you need for an empty project is to create a helloworld.cpp with your favorite text editor.

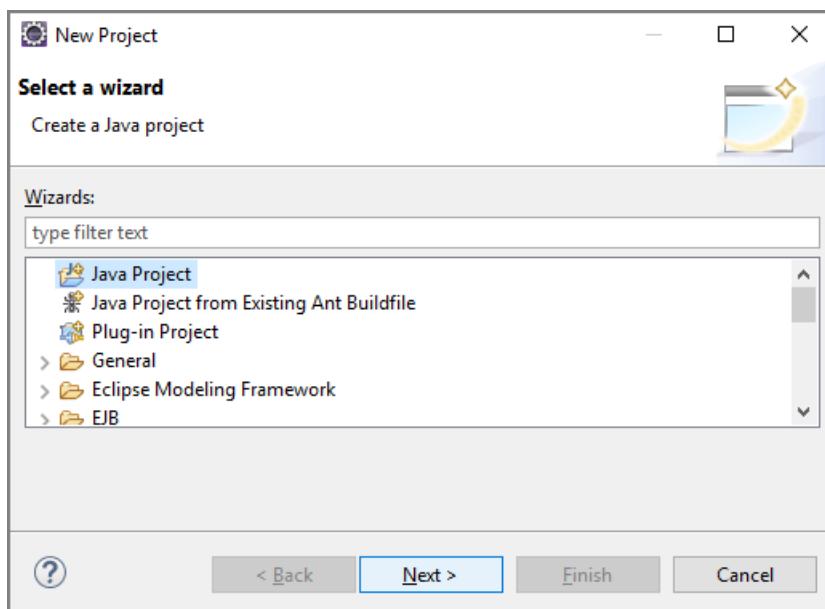
## Choose your target environment

- Java Runtime
- Android

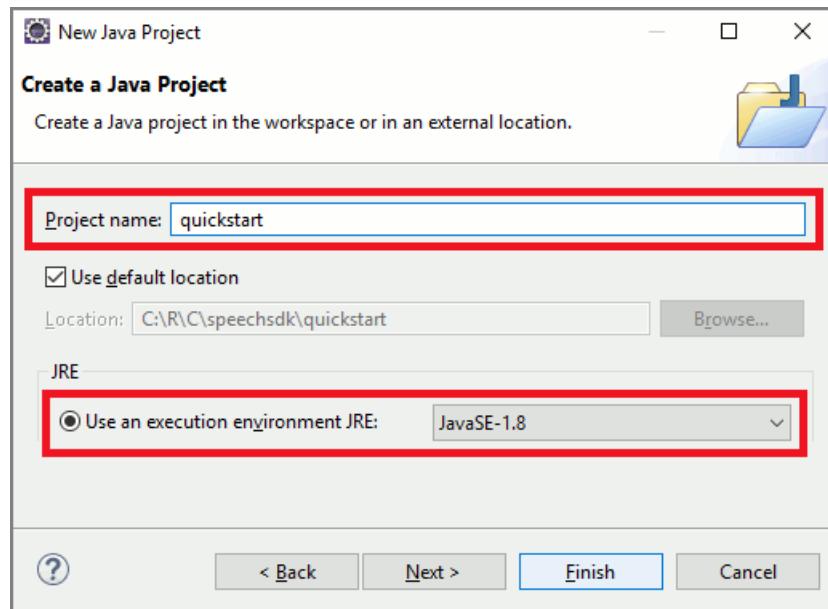
1. Start Eclipse.
2. In the Eclipse Launcher, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.



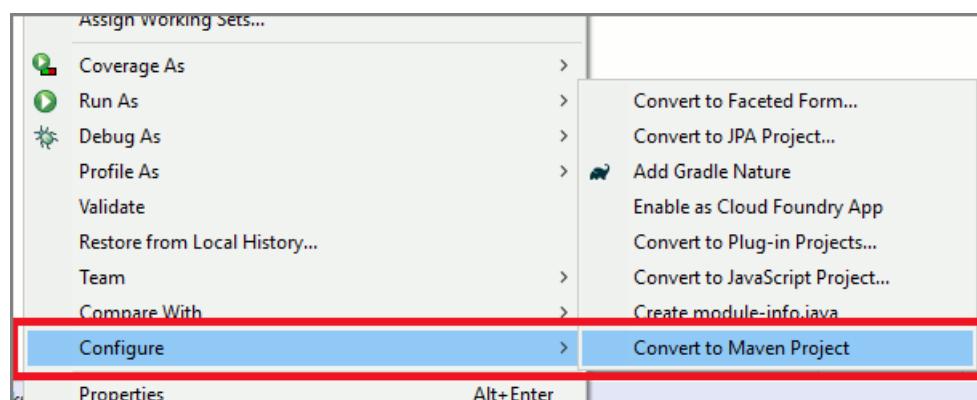
3. In a moment, the main window of the Eclipse IDE appears. Close the **Welcome** screen if one is present.
4. From the Eclipse menu bar, create a new project by choosing **File > New > Project**.
5. The **New Project** dialog box appears. Select **Java Project**, and select **Next**.



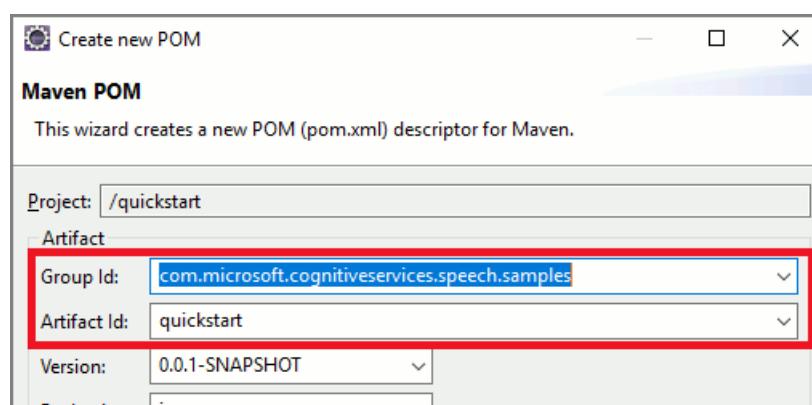
6. The **New Java Project** wizard starts. In the **Project name** field, enter **quickstart**, and choose **JavaSE-1.8** as the execution environment. Select **Finish**.



7. If the **Open Associated Perspective?** window appears, select **Open Perspective**.
8. In the **Package explorer**, right-click the **quickstart** project. Choose **Configure > Convert to Maven Project** from the context menu.



9. The **Create new POM** window appears. In the **Group Id** field, enter `com.microsoft.cognitiveservices.speech.samples`, and in the **Artifact Id** field, enter `quickstart`. Then select **Finish**.



10. Open the `pom.xml` file and edit it.
  - At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK, as shown here:

```
<repositories>
 <repository>
 <id>maven-cognitiveservices-speech</id>
 <name>Microsoft Cognitive Services Speech Maven Repository</name>
 <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
 </repository>
</repositories>
```

- Also add a `dependencies` element, with the Speech SDK version 1.7.0 as a dependency:

```
<dependencies>
 <dependency>
 <groupId>com.microsoft.cognitiveservices.speech</groupId>
 <artifactId>client-sdk</artifactId>
 <version>1.7.0</version>
 </dependency>
</dependencies>
```

- Save the changes.

For the Python-based quickstarts, all you'll need to do is create a file named `helloworld.py` with your favorite text editor or IDE.

## Next steps

- [Quickstart: Recognize speech from a microphone](#)
- [Quickstart: Recognize speech from a file](#)
- [Quickstart: Translate speech-to-text](#)
- [Quickstart: Synthesize speech to text](#)
- [Quickstart: Recognize Intents](#)

# Quickstart: Setup development environment

11/20/2019 • 25 minutes to read • [Edit Online](#)

## Choose your target environment

- [.NET](#)
- [.NET Core](#)
- [Unity](#)
- [UWP](#)
- [Xamarin](#)

This guide shows how to install the [Speech SDK](#) for .NET Framework (Windows).

### IMPORTANT

By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

## Prerequisites

This quickstart requires:

- [Visual Studio 2019](#)

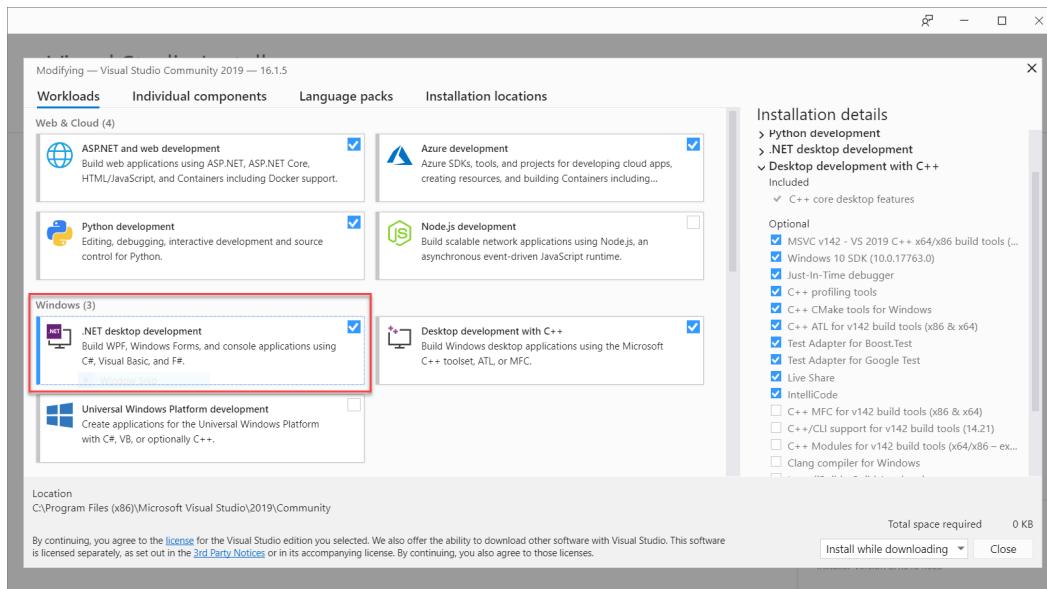
## Create a Visual Studio project and install the Speech SDK

You'll need to install the [Speech SDK NuGet package](#) so you can reference it in your code. To do that, you may first need to create a **helloworld** project. If you already have a project with the **.NET desktop development** workload available, you can use that project and skip to [Use NuGet Package Manager to install the Speech SDK](#).

### Create helloworld project

1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.
3. In the **Create a new project** window, choose **Console App (.NET Framework)**, and then select **Next**.
4. In the **Configure your new project** window, enter *helloworld* in **Project name**, choose or create the directory path in **Location**, and then select **Create**.
5. From the Visual Studio menu bar, select **Tools > Get Tools and Features**, which opens Visual Studio Installer and displays the **Modifying** dialog box.
6. Check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, select the check box next to it, and then select **Modify** to start the installation. It may take a few minutes to download and install.

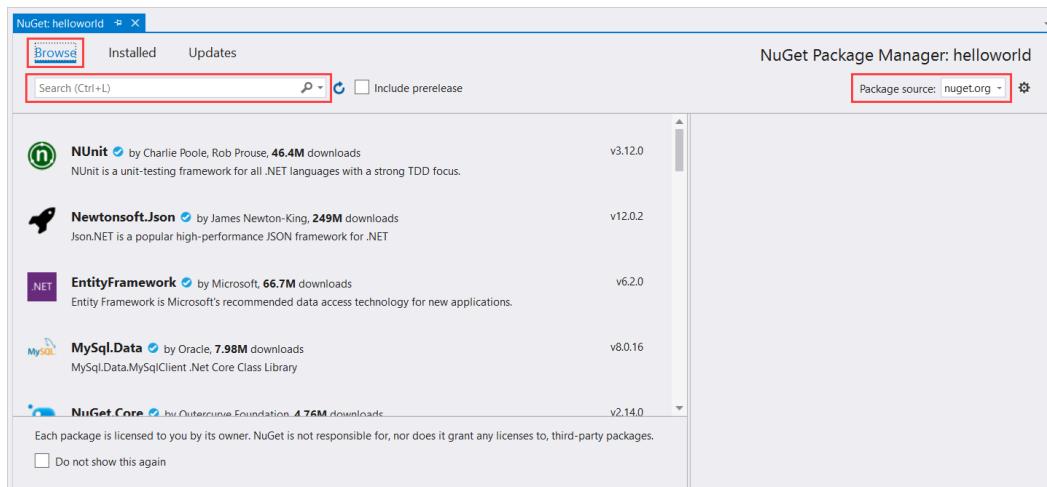
If the check box next to **.NET desktop development** is already selected, select **Close** to exit the dialog box.



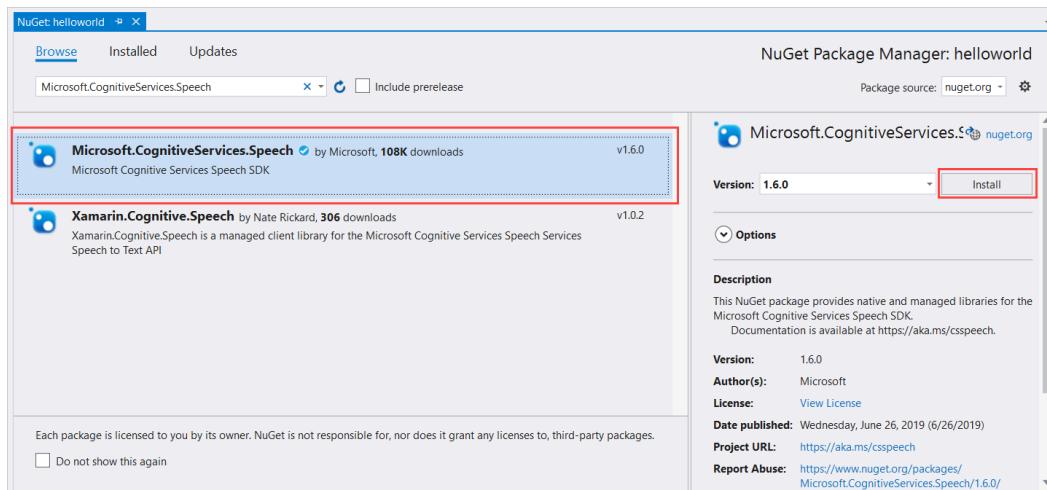
## 7. Close Visual Studio Installer.

### Use NuGet Package Manager to install the Speech SDK

- In the Solution Explorer, right-click the **helloworld** project, and then select **Manage NuGet Packages** to show the NuGet Package Manager.



- In the upper-right corner, find the **Package Source** drop-down box, and make sure that `nuget.org` is selected.
- In the upper-left corner, select **Browse**.
- In the search box, type `Microsoft.CognitiveServices.Speech` and select **Enter**.
- From the search results, select the **Microsoft.CognitiveServices.Speech** package, and then select **Install** to install the latest stable version.



## 6. Accept all agreements and licenses to start the installation.

After the package is installed, a confirmation appears in the **Package Manager Console** window.

You can now move on to [Next steps](#) below.

## Next steps

- [Quickstart: Recognize speech from a microphone](#)
- [Quickstart: Recognize speech from a file](#)
- [Quickstart: Recognize speech from an Azure Blob](#)
- [Quickstart: Translate speech-to-text](#)
- [Quickstart: Synthesize text to an audio device](#)
- [Quickstart: Synthesize text to a file](#)
- [Quickstart: Recognize Intents](#)

### Choose your target environment

- [Linux](#)
- [macOS](#)
- [Windows](#)

This guide shows how to install the [Speech SDK](#) for Linux

#### IMPORTANT

By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

## System requirements

Linux (Ubuntu 16.04, Ubuntu 18.04, Debian 9)

## Prerequisites

To complete this quickstart, you'll need:

- [Visual Studio 2019](#)
- Supported Linux platforms will require certain libraries installed (`libssl` for secure sockets)

layer support and `libasound2` for sound support). Refer to your distribution below for the commands needed to install the correct versions of these libraries.

- On Ubuntu:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libasound2 wget
```

- On Debian 9:

```
sudo apt-get update
sudo apt-get install build-essential libssl1.0.2 libasound2 wget
```

## Install Speech SDK

The Speech SDK for Linux can be used to build both 64-bit and 32-bit applications. The required libraries and header files can be downloaded as a tar file from <https://aka.ms/csspeech/linuxbinary>.

Download and install the SDK as follows:

1. Choose a directory to which the Speech SDK files should be extracted, and set the `SPEECHSDK_ROOT` environment variable to point to that directory. This variable makes it easy to refer to the directory in future commands. For example, if you want to use the directory `speechsdk` in your home directory, use a command like the following:

```
export SPEECHSDK_ROOT="$HOME/speechsdk"
```

2. Create the directory if it doesn't exist yet.

```
mkdir -p "$SPEECHSDK_ROOT"
```

3. Download and extract the `.tar.gz` archive containing the Speech SDK binaries:

```
wget -O SpeechSDK-Linux.tar.gz https://aka.ms/csspeech/linuxbinary
tar --strip 1 -xzf SpeechSDK-Linux.tar.gz -C "$SPEECHSDK_ROOT"
```

4. Validate the contents of the top-level directory of the extracted package:

```
ls -l "$SPEECHSDK_ROOT"
```

The directory listing should contain the third-party notice and license files, as well as an `include` directory containing header (`.h`) files and a `lib` directory containing libraries.

PATH	DESCRIPTION
<code>license.md</code>	License
<code>ThirdPartyNotices.md</code>	Third-party notices.
<code>REDIST.txt</code>	Redistribution notice.

PATH	DESCRIPTION
include	The required header files for C and C++
lib/x64	Native library for x64 required to link your application
lib/x86	Native library for x86 required to link your application

You can now move on to [Next steps](#) below.

## Next steps

- [Quickstart: Recognize speech from a microphone](#)
- [Quickstart: Recognize speech from a file](#)
- [Quickstart: Recognize speech from an Azure Blob](#)
- [Quickstart: Translate speech-to-text](#)
- [Quickstart: Synthesize text to an audio device](#)
- [Quickstart: Synthesize text to a file](#)
- [Quickstart: Recognize Intents](#)

### Choose your target environment

- [Java Runtime](#)
- [Android](#)

This guide shows how to install the [Speech SDK](#) for 64-bit Java 8 JRE.

#### NOTE

For the Speech Devices SDK and the Roobo device, see [Speech Devices SDK](#).

#### IMPORTANT

By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

## Supported operating systems

- The Java Speech SDK package is available for these operating systems:
  - Windows: 64-bit only
  - Mac: macOS X version 10.13 or later
  - Linux: 64-bit only on Ubuntu 16.04, Ubuntu 18.04, or Debian 9

## Prerequisites

- [Java 8 or JDK 8](#)
- [Eclipse Java IDE](#) (requires Java already installed)
- Supported Linux platforms will require certain libraries installed ( `libssl` for secure sockets

layer support and `libasound2` for sound support). Refer to your distribution below for the commands needed to install the correct versions of these libraries.

- On Ubuntu, run the following commands to install the required packages:

```
```sh
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libasound2
```
```

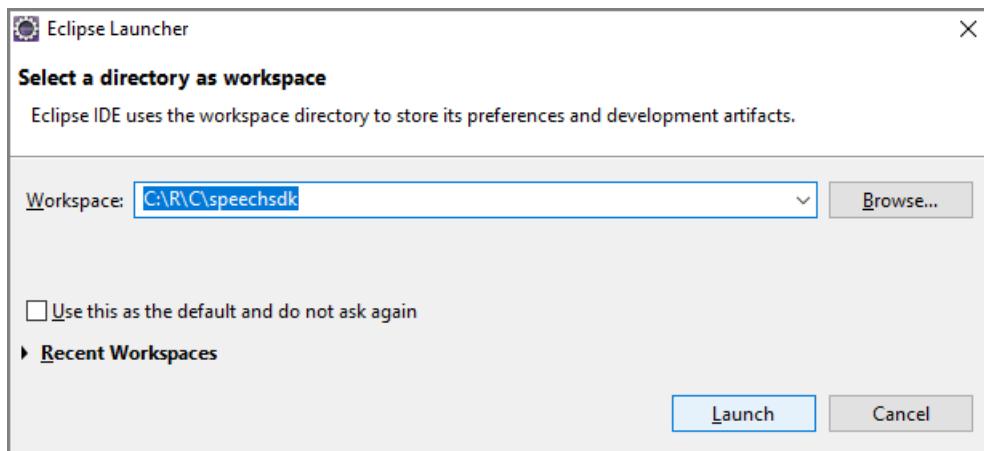
- On Debian 9, run the following commands to install the required packages:

```
```sh
sudo apt-get update
sudo apt-get install build-essential libssl1.0.2 libasound2
```
```

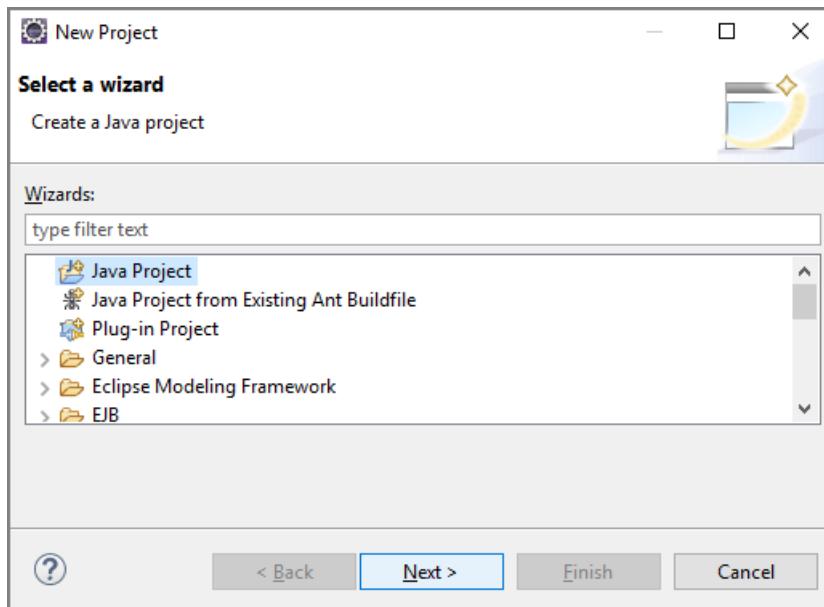
- On Windows, you need the [Microsoft Visual C++ Redistributable for Visual Studio 2019](#) for your platform. Note that installing this for the first time may require you to restart Windows before continuing with this guide.

## Create an Eclipse project and install the Speech SDK

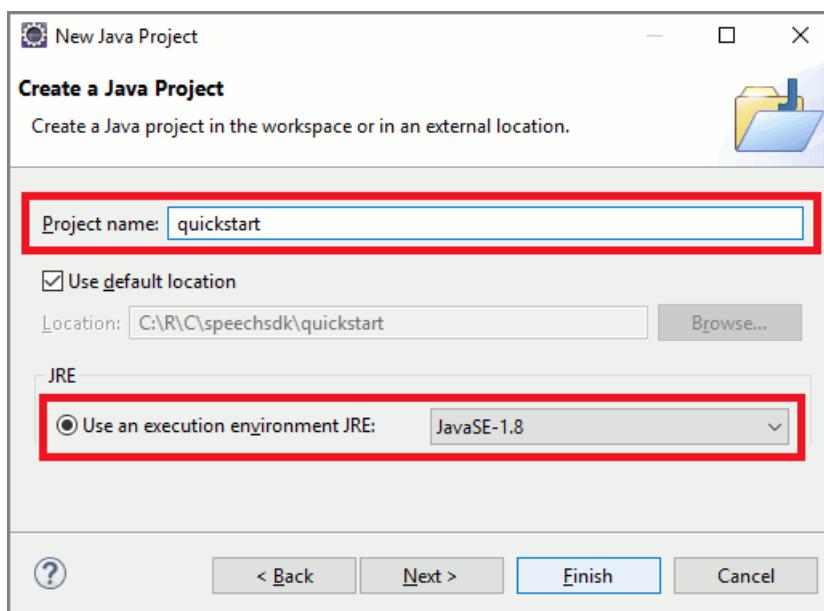
1. Start Eclipse.
2. In the Eclipse Launcher, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.



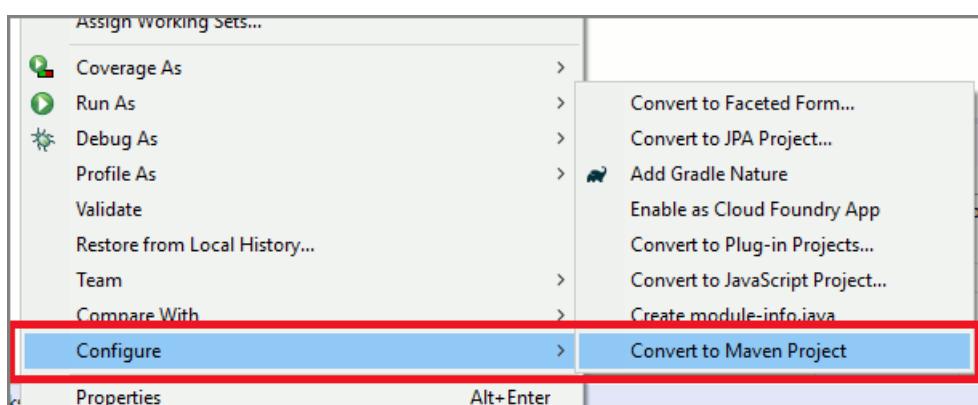
3. In a moment, the main window of the Eclipse IDE appears. Close the **Welcome** screen if one is present.
4. From the Eclipse menu bar, create a new project by choosing **File > New > Project**.
5. The **New Project** dialog box appears. Select **Java Project**, and select **Next**.



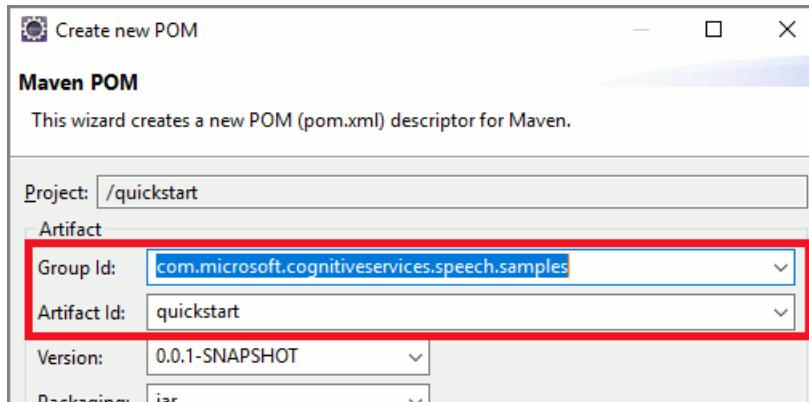
6. The **New Java Project** wizard starts. In the **Project name** field, enter **quickstart**, and choose **JavaSE-1.8** as the execution environment. Select **Finish**.



7. If the **Open Associated Perspective?** window appears, select **Open Perspective**.
8. In the **Package explorer**, right-click the **quickstart** project. Choose **Configure > Convert to Maven Project** from the context menu.



9. The **Create new POM** window appears. In the **Group Id** field, enter **com.microsoft.cognitiveservices.speech.samples**, and in the **Artifact Id** field, enter **quickstart**. Then select **Finish**.



10. Open the `pom.xml` file and edit it.

- At the end of the file, before the closing tag `</project>`, create a `repositories` element with a reference to the Maven repository for the Speech SDK, as shown here:

```
<repositories>
 <repository>
 <id>maven-cognitiveservices-speech</id>
 <name>Microsoft Cognitive Services Speech Maven Repository</name>
 <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
 </repository>
</repositories>
```

- Also add a `dependencies` element, with the Speech SDK version 1.7.0 as a dependency:

```
<dependencies>
 <dependency>
 <groupId>com.microsoft.cognitiveservices.speech</groupId>
 <artifactId>client-sdk</artifactId>
 <version>1.7.0</version>
 </dependency>
</dependencies>
```

- Save the changes.

## Next steps

- [Quickstart: Recognize speech from a microphone](#)
- [Quickstart: Recognize speech from a file](#)
- [Quickstart: Recognize speech from an Azure Blob](#)
- [Quickstart: Translate speech-to-text](#)
- [Quickstart: Synthesize text to an audio device](#)
- [Quickstart: Synthesize text to a file](#)
- [Quickstart: Recognize Intents](#)

This guide shows how to install the [Speech SDK](#) for Python.

### IMPORTANT

By downloading any of the Speech SDK for Azure Cognitive Services components on this page, you acknowledge its license. See the [Microsoft Software License Terms for the Speech SDK](#).

## Supported operating systems

- The Python Speech SDK package is available for these operating systems:
  - Windows: x64 and x86
  - Mac: macOS X version 10.12 or later
  - Linux: Ubuntu 16.04, Ubuntu 18.04, Debian 9 on x64

## Prerequisites

- Supported Linux platforms will require certain libraries installed (`libssl` for secure sockets layer support and `libasound2` for sound support). Refer to your distribution below for the commands needed to install the correct versions of these libraries.
  - On Ubuntu, run the following commands to install the required packages:

```
```sh
sudo apt-get update
sudo apt-get install build-essential libssl1.0.0 libasound2
````
```

- On Debian 9, run the following commands to install the required packages:

```
```sh
sudo apt-get update
sudo apt-get install build-essential libssl1.0.2 libasound2
````
```

- On Windows, you need the [Microsoft Visual C++ Redistributable for Visual Studio 2019](#) for your platform. Note that installing this for the first time may require you to restart Windows before continuing with this guide.
- And finally, you'll need [Python 3.5, 3.6 or 3.7](#). To check your installation, open a command prompt and type the command `python --version` and check the result. If it's installed properly, you'll get a response "Python 3.5.1" or similar.

## Install the Speech SDK using Visual Studio Code

1. Download and install the latest supported version of [Python](#) for your platform, 3.5 or later.
  - Windows users make sure to select "Add Python to your PATH" during the installation process.
2. Download and install [Visual Studio Code](#).
3. Open Visual Studio Code and install the Python extension. Select **File > Preferences > Extensions** from the menu. Search for **Python** and click **Install**.



4. Also from within Visual Studio Code, install the Speech SDK Python package from the integrated command line:
  - a. Open a terminal (from the drop-down menus, **Terminal** > **New Terminal**)
  - b. In the terminal that opens, enter the command

```
python -m pip install azure-cognitiveservices-speech
```

That's it, you're ready to start coding to the Speech SDK in Python, and can move on to [Next steps](#) below. If you are new to Visual Studio Code, refer to the more extensive [Visual Studio Code Documentation](#). For more information about Visual Studio Code and Python, see [Visual Studio Code Python tutorial](#).

## Install the Speech SDK using the command line

If you are not using Visual Studio Code, the following command installs the Python package from [PyPI](#) for the Speech SDK. For users of Visual Studio Code, skip to the next sub-section.

```
pip install azure-cognitiveservices-speech
```

If you are on macOS, you may need to run the following command to get the `pip` command above to work:

```
python3 -m pip install --upgrade pip
```

Once you've successfully used `pip` to install `azure-cognitiveservices-speech`, you can use the Speech SDK by importing the namespace into your Python projects. For example:

```
import azure.cognitiveservices.speech as speechsdk
```

This is shown in more detail within the code examples listed in [Next steps](#) below.

## Support and updates

Updates to the Speech SDK Python package are distributed via PyPI and announced in the [Release notes](#). If a new version is available, you can update to it with the command

`pip install --upgrade azure-cognitiveservices-speech`. Check which version is currently installed by inspecting the `azure.cognitiveservices.speech.__version__` variable.

If you have a problem, or you're missing a feature, see [Support and help options](#).

## Next steps

- [Quickstart: Recognize speech from a microphone](#)
- [Quickstart: Recognize speech from a file](#)
- [Quickstart: Recognize speech from an Azure Blob](#)
- [Quickstart: Translate speech-to-text](#)
- [Quickstart: Synthesize text to an audio device](#)
- [Quickstart: Synthesize text to a file](#)
- [Quickstart: Recognize Intents](#)

# Enable logging in the Speech SDK

12/4/2019 • 2 minutes to read • [Edit Online](#)

Logging to file is an optional feature for the Speech SDK. During development logging provides additional information and diagnostics from the Speech SDK's core components. It can be enabled by setting the property `Speech_LogFilename` on a speech configuration object to the location and name of the log file. Logging will be activated globally once a recognizer is created from that configuration and can't be disabled afterwards. You can't change the name of a log file during a running logging session.

## NOTE

Logging is available since Speech SDK version 1.4.0 in all supported Speech SDK programming languages, with the exception of JavaScript.

## Sample

The log file name is specified on a configuration object. Taking the `SpeechConfig` as an example and assuming that you have created an instance called `config`:

```
config SetProperty(PropertyId.Speech_LogFilename, "LogfilePathAndName");
```

```
config.setProperty(PropertyId.Speech_LogFilename, "LogfilePathAndName");
```

```
config->SetProperty(PropertyId::Speech_LogFilename, "LogfilePathAndName");
```

```
config.set_property(speechsdk.PropertyId.Speech_LogFilename, "LogfilePathAndName")
```

```
[config SetPropertyTo:@"LogfilePathAndName" byId:SPXSpeechLogFilename];
```

You can create a recognizer from the config object. This will enable logging for all recognizers.

## NOTE

If you create a `SpeechSynthesizer` from the config object, it will not enable logging. If logging is enabled though, you will also receive diagnostics from the `SpeechSynthesizer`.

## Create a log file on different platforms

For Windows or Linux, the log file can be in any path the user has write permission for. Write permissions to file system locations in other operating systems may be limited or restricted by default.

### Universal Windows Platform (UWP)

UWP applications need to place log files in one of the application data locations (local, roaming, or temporary). A log file can be created in the local application folder:

```
StorageFolder storageFolder = ApplicationData.Current.LocalFolder;
StorageFile logFile = await storageFolder.CreateFileAsync("logfile.txt",
CreationCollisionOption.ReplaceExisting);
config SetProperty(PropertyId.Speech_LogFilename, logFile.Path);
```

More about file access permission for UWP applications is available [here](#).

## Android

You can save a log file to either internal storage, external storage, or the cache directory. Files created in the internal storage or the cache directory are private to the application. It is preferable to create a log file in external storage.

```
File dir = context.getExternalFilesDir(null);
File logFile = new File(dir, "logfile.txt");
config.setProperty(PropertyId.Speech_LogFilename, logFile.getAbsolutePath());
```

The code above will save a log file to the external storage in the root of an application-specific directory. A user can access the file with the file manager (usually in `Android/data/ApplicationName/logfile.txt`). The file will be deleted when the application is uninstalled.

You also need to request `WRITE_EXTERNAL_STORAGE` permission in the manifest file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="...>
...
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
...
</manifest>
```

More about data and file storage for Android applications is available [here](#).

## iOS

Only directories inside the application sandbox are accessible. Files can be created in the documents, library, and temp directories. Files in the documents directory can be made available to a user. The following code snippet shows creation of a log file in the application document directory:

```
NSString *filePath = [
 [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) firstObject]
 stringByAppendingPathComponent:@"logfile.txt"];
[speechConfig SetPropertyTo:filePath byId:SPXSpeechLogFilename];
```

To access a created file, add the below properties to the `Info.plist` property list of the application:

```
<key>UIFileSharingEnabled</key>
<true/>
<key>LSSupportsOpeningDocumentsInPlace</key>
<true/>
```

More about iOS File System is available [here](#).

## Next steps

Explore our samples on [GitHub](#)

# How to track Speech SDK memory usage

12/11/2019 • 2 minutes to read • [Edit Online](#)

The Speech SDK is based on a native code base that's projected into multiple programming languages through a series of interoperability layers. Each language-specific projection has idiomatically correct features to manage the object lifecycle. Additionally, the Speech SDK includes memory management tooling to track resource usage with object logging and object limits.

## How to read object logs

If [Speech SDK logging is enabled](#), tracking tags are emitted to enable historical object observation. These tags include:

- `TrackHandle` or `StopTracking`
- The object type
- The current number of objects that are tracked the type of the object, and the current number being tracked.

Here's a sample log:

```
(284): 8604ms SPX_DBG_TRACE_VERBOSE: handle_table.h:90 TrackHandle
type=Microsoft::CognitiveServices::Speech::Impl::ISpxRecognitionResult handle=0x0x7f688401e1a0,
ptr=0x0x7f688401e1a0, total=19
```

## Set a warning threshold

You have the option to create a warning threshold, and if that threshold is exceeded (assuming logging is enabled), a warning message is logged. The warning message contains a dump of all objects in existence along with their count. This information can be used to better understand issues.

To enable a warning threshold, it must be specified on a `SpeechConfig` object. This object is checked when a new recognizer is created. In the following examples, let's assume that you've created an instance of `SpeechConfig` called `config`:

```
config SetProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

```
config->SetProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

```
config.setProperty("SPEECH-ObjectCountWarnThreshold", "10000");
```

```
speech_config.set_property_by_name("SPEECH-ObjectCountWarnThreshold", "10000")?
```

```
[config SetPropertyTo:@"10000" byName:"SPEECH-ObjectCountWarnThreshold"];
```

**TIP**

The default value for this property is 10,000.

## Set an error threshold

Using the Speech SDK, you can set the maximum number of objects allowed at a given time. If this setting is enabled, when the maximum number is hit, attempts to create new recognizer objects will fail. Existing objects will continue to work.

Here's a sample error:

```
Runtime error: The maximum object count of 500 has been exceeded.
The threshold can be adjusted by setting the SPEECH-ObjectCountErrorThreshold property on the SpeechConfig
object.
See https://docs.microsoft.com/azure/cognitive-services/speech-service/how-to-object-tracking-speech-sdk
for more detailed information.
Handle table dump by object type:
class Microsoft::CognitiveServices::Speech::Impl::ISpxRecognitionResult 0
class Microsoft::CognitiveServices::Speech::Impl::ISpxRecognizer 0
class Microsoft::CognitiveServices::Speech::Impl::ISpxAudioConfig 0
class Microsoft::CognitiveServices::Speech::Impl::ISpxSpeechConfig 0
```

To enable an error threshold, it must be specified on a `SpeechConfig` object. This object is checked when a new recognizer is created. In the following examples, let's assume that you've created an instance of `SpeechConfig` called `config`:

```
config SetProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

```
config->SetProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

```
config.setProperty("SPEECH-ObjectCountErrorThreshold", "10000");
```

```
speech_config.set_property_by_name("SPEECH-ObjectCountErrorThreshold", "10000")?
```

```
[config SetPropertyTo:@"10000" byName:"SPEECH-ObjectCountErrorThreshold"];
```

**TIP**

The default value for this property is the platform-specific maximum value for a `size_t` data type. A typical recognition will consume between 7 and 10 internal objects.

## Next steps

- [Get your Speech service trial subscription](#)
- [Learn how to recognize speech using a microphone](#)

# Release notes

12/16/2019 • 14 minutes to read • [Edit Online](#)

## Speech SDK 1.8.0: 2019-November release

### New Features

- Added a `FromHost()` API, to ease use with on-prem containers and sovereign clouds.
- Added Automatic Source Language Detection for Speech Recognition (in Java and C++)
- Added `SourceLanguageConfig` object for Speech Recognition, used to specify expected source languages (in Java and C++)
- Added `KeywordRecognizer` support on Windows (UWP), Android and iOS through the Nuget and Unity packages
- Added Remote Conversation Java API to do Conversation Transcription in asynchronous batches.

### Breaking changes

- Conversation Transcriber functionalities moved under namespace `Microsoft.CognitiveServices.Speech.Transcription`.
- Part of the Conversation Transcriber methods are moved to new `Conversation` class.
- Dropped support for 32-bit (ARMv7 and x86) iOS

### Bug fixes

- Fix for crash if local `KeywordRecognizer` is used without a valid speech service subscription key

### Samples

- Xamarin sample for `KeywordRecognizer`
- Unity sample for `KeywordRecognizer`
- C++ and Java samples for Automatic Source Language Detection.

## Speech SDK 1.7.0: 2019-September release

### New Features

- Added beta support for Xamarin on Universal Windows Platform (UWP), Android, and iOS
- Added iOS support for Unity
- Added `compressed` input support for ALaw, Mulaw, FLAC on Android, iOS and Linux
- Added `SendMessageAsync` in `Connection` class for sending a message to service
- Added `SetMessageProperty` in `Connection` class for setting property of a message
- TTS added bindings for Java (Jre and Android), Python, Swift, and Objective-C
- TTS added playback support for macOS, iOS, and Android.
- Added "word boundary" information for TTS.

### Bug fixes

- Fixed IL2CPP build issue on Unity 2019 for Android
- Fixed issue with malformed headers in wav file input being processed incorrectly
- Fixed issue with UUIDs not being unique in some connection properties

- Fixed a few warnings about nullability specifiers in the Swift bindings (might require small code changes)
- Fixed a bug that caused websocket connections to be closed ungracefully under network load
- Fixed an issue on Android that sometimes results in duplicate impression IDs used by `DialogServiceConnector`
- Improvements to the stability of connections across multi-turn interactions and the reporting of failures (via `Canceled` events) when they occur with `DialogServiceConnector`
- `DialogServiceConnector` session starts will now properly provide events, including when calling `ListenOnceAsync()` during an active `StartKeywordRecognitionAsync()`
- Addressed a crash associated with `DialogServiceConnector` activities being received

## Samples

- Quickstart for Xamarin
- Updated CPP Quickstart with Linux ARM64 information
- Updated Unity quickstart with iOS information

# Speech SDK 1.6.0: 2019-June release

## Samples

- Quickstart samples for Text To Speech on UWP and Unity
- Quickstart sample for Swift on iOS
- Unity samples for Speech & Intent Recognition and Translation
- Updated quickstart samples for `DialogServiceConnector`

## Improvements / Changes

- Dialog namespace:
  - `SpeechBotConnector` has been renamed to `DialogServiceConnector`
  - `BotConfig` has been renamed to `DialogServiceConfig`
  - `BotConfig::FromChannelSecret()` has been remapped to `DialogServiceConfig::FromBotSecret()`
  - All existing Direct Line Speech clients continue to be supported after the rename
- Update TTS REST adapter to support proxy, persistent connection
- Improve error message when an invalid region is passed
- Swift/Objective-C:
  - Improved error reporting: Methods that can result in an error are now present in two versions: One that exposes an `NSError` object for error handling, and one that raises an exception. The former are exposed to Swift. This change requires adaptations to existing Swift code.
  - Improved event handling

## Bug fixes

- Fix for TTS: where `SpeakTextAsync` future returned without waiting until audio has completed rendering
- Fix for marshaling strings in C# to enable full language support
- Fix for .NET core app problem to load core library with net461 target framework in samples
- Fix for occasional issues to deploy native libraries to the output folder in samples
- Fix for web socket closing reliably
- Fix for possible crash while opening a connection under very heavy load on Linux
- Fix for missing metadata in the framework bundle for macOS
- Fix for problems with `pip install --user` on Windows

## Speech SDK 1.5.1

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

### Bug fixes

- Fix FromSubscription when used with Conversation Transcription.
- Fix bug in keyword spotting for voice assistants.

## Speech SDK 1.5.0: 2019-May release

### New features

- Keyword spotting (KWS) is now available for Windows and Linux. KWS functionality might work with any microphone type, official KWS support, however, is currently limited to the microphone arrays found in the Azure Kinect DK hardware or the Speech Devices SDK.
- Phrase hint functionality is available through the SDK. For more information, see [here](#).
- Conversation transcription functionality is available through the SDK. See [here](#).
- Add support for voice assistants using the Direct Line Speech channel.

### Samples

- Added samples for new features or new services supported by the SDK.

### Improvements / Changes

- Added various recognizer properties to adjust service behavior or service results (like masking profanity and others).
- You can now configure the recognizer through the standard configuration properties, even if you created the recognizer `FromEndpoint`.
- Objective-C: `OutputFormat` property was added to `SPXSpeechConfiguration`.
- The SDK now supports Debian 9 as a Linux distribution.

### Bug fixes

- Fixed a problem where the speaker resource was destructed too early in text-to-speech.

## Speech SDK 1.4.2

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

## Speech SDK 1.4.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Prevent web pack from loading https-proxy-agent.

## Speech SDK 1.4.0: 2019-April release

### New features

- The SDK now supports the text-to-speech service as a beta version. It is supported on Windows and Linux Desktop from C++ and C#. For more information, check the [text-to-speech overview](#).
- The SDK now supports MP3 and Opus/OGG audio files as stream input files. This feature is available only

on Linux from C++ and C# and is currently in beta (more details [here](#)).

- The Speech SDK for Java, .NET core, C++ and Objective-C have gained macOS support. The Objective-C support for macOS is currently in beta.
- iOS: The Speech SDK for iOS (Objective-C) is now also published as a CocoaPod.
- JavaScript: Support for non-default microphone as an input device.
- JavaScript: Proxy support for Node.js.

## Samples

- Samples for using the Speech SDK with C++ and with Objective-C on macOS have been added.
- Samples demonstrating the usage of the text-to-speech service have been added.

## Improvements / Changes

- Python: Additional properties of recognition results are now exposed via the `properties` property.
- For additional development and debug support, you can redirect SDK logging and diagnostics information into a log file (more details [here](#)).
- JavaScript: Improve audio processing performance.

## Bug fixes

- Mac/iOS: A bug that led to a long wait when a connection to the Speech service could not be established was fixed.
- Python: improve error handling for arguments in Python callbacks.
- JavaScript: Fixed wrong state reporting for speech ended on RequestSession.

# Speech SDK 1.3.1: 2019-February refresh

This is a bug fix release and only affecting the native/managed SDK. It is not affecting the JavaScript version of the SDK.

## Bug fix

- Fixed a memory leak when using microphone input. Stream based or file input is not affected.

# Speech SDK 1.3.0: 2019-February release

## New Features

- The Speech SDK supports selection of the input microphone through the `AudioConfig` class. This allows you to stream audio data to the Speech service from a non-default microphone. For more information, see the documentation describing [audio input device selection](#). This feature is not yet available from JavaScript.
- The Speech SDK now supports Unity in a beta version. Provide feedback through the issue section in the [GitHub sample repository](#). This release supports Unity on Windows x86 and x64 (desktop or Universal Windows Platform applications), and Android (ARM32/64, x86). More information is available in our [Unity quickstart](#).
- The file `Microsoft.CognitiveServices.Speech.csharp.bindings.dll` (shipped in previous releases) isn't needed anymore. The functionality is now integrated into the core SDK.

## Samples

The following new content is available in our [sample repository](#):

- Additional samples for `AudioConfig.FromMicrophoneInput`.
- Additional Python samples for intent recognition and translation.

- Additional samples for using the `Connection` object in iOS.
- Additional Java samples for translation with audio output.
- New sample for use of the [Batch Transcription REST API](#).

## Improvements / Changes

- Python
  - Improved parameter verification and error messages in `SpeechConfig`.
  - Add support for the `Connection` object.
  - Support for 32-bit Python (x86) on Windows.
  - The Speech SDK for Python is out of beta.
- iOS
  - The SDK is now built against the iOS SDK version 12.1.
  - The SDK now supports iOS versions 9.2 and later.
  - Improve reference documentation and fix several property names.
- JavaScript
  - Add support for the `Connection` object.
  - Add type definition files for bundled JavaScript
  - Initial support and implementation for phrase hints.
  - Return properties collection with service JSON for recognition
- Windows DLLs do now contain a version resource.
- If you create a recognizer `FromEndpoint` you can add parameters directly to the endpoint URL. Using `FromEndpoint` you can't configure the recognizer through the standard configuration properties.

## Bug fixes

- Empty proxy username and proxy password were not handled correctly. With this release, if you set proxy username and proxy password to an empty string, they will not be submitted when connecting to the proxy.
- SessionId's created by the SDK were not always truly random for some languages / environments. Added random generator initialization to fix this issue.
- Improve handling of authorization token. If you want to use an authorization token, specify in the `SpeechConfig` and leave the subscription key empty. Then create the recognizer as usual.
- In some cases the `Connection` object wasn't released correctly. This issue has been fixed.
- The JavaScript sample was fixed to support audio output for translation synthesis also on Safari.

## Speech SDK 1.2.1

This is a JavaScript-only release. No features have been added. The following fixes were made:

- Fire end of stream at `turn.end`, not at `speech.end`.
- Fix bug in audio pump that did not schedule next send if the current send failed.
- Fix continuous recognition with auth token.
- Bug fix for different recognizer / endpoints.
- Documentation improvements.

## Speech SDK 1.2.0: 2018-December release

### New Features

- Python

- The Beta version of Python support (3.5 and above) is available with this release. For more information, see [here](#)](quickstart-python.md).
- JavaScript
  - The Speech SDK for JavaScript has been open-sourced. The source code is available on [GitHub](#).
  - We now support Node.js, more info can be found [here](#).
  - The length restriction for audio sessions has been removed, reconnection will happen automatically under the cover.
- `Connection` object
  - From the `Recognizer`, you can access a `Connection` object. This object allows you to explicitly initiate the service connection and subscribe to connect and disconnect events. (This feature is not yet available from JavaScript and Python.)
- Support for Ubuntu 18.04.
- Android
  - Enabled ProGuard support during APK generation.

## Improvements

- Improvements in the internal thread usage, reducing the number of threads, locks, mutexes.
- Improved error reporting / information. In several cases, error messages have not been propagated out all the way out.
- Updated development dependencies in JavaScript to use up-to-date modules.

## Bug fixes

- Fixed memory leaks due to a type mismatch in `RecognizeAsync`.
- In some cases exceptions were being leaked.
- Fixing memory leak in translation event arguments.
- Fixed a locking issue on reconnect in long running sessions.
- Fixed an issue that could lead to missing final result for failed translations.
- C#: If an `async` operation wasn't awaited in the main thread, it was possible the recognizer could be disposed before the async task was completed.
- Java: Fixed a problem resulting in a crash of the Java VM.
- Objective-C: Fixed enum mapping; `RecognizedIntent` was returned instead of `RecognizingIntent`.
- JavaScript: Set default output format to 'simple' in `SpeechConfig`.
- JavaScript: Removing inconsistency between properties on the config object in JavaScript and other languages.

## Samples

- Updated and fixed several samples (for example output voices for translation, etc.).
- Added Node.js samples in the [sample repository](#).

# Speech SDK 1.1.0

## New Features

- Support for Android x86/x64.
- Proxy Support: In the `SpeechConfig` object, you can now call a function to set the proxy information (hostname, port, username, and password). This feature is not yet available on iOS.
- Improved error code and messages. If a recognition returned an error, this did already set `Reason` (in canceled event) or `CancellationDetails` (in recognition result) to `Error`. The canceled event now contains two additional members, `ErrorCode` and `ErrorDetails`. If the server returned additional error information

with the reported error, it will now be available in the new members.

## Improvements

- Added additional verification in the recognizer configuration, and added additional error message.
- Improved handling of long-time silence in middle of an audio file.
- NuGet package: for .NET Framework projects, it prevents building with AnyCPU configuration.

## Bug fixes

- Fixed several exceptions found in recognizers. In addition, exceptions are caught and converted into `Canceled` event.
- Fix a memory leak in property management.
- Fixed bug in which an audio input file could crash the recognizer.
- Fixed a bug where events could be received after a session stop event.
- Fixed some race conditions in threading.
- Fixed an iOS compatibility issue that could result in a crash.
- Stability improvements for Android microphone support.
- Fixed a bug where a recognizer in JavaScript would ignore the recognition language.
- Fixed a bug preventing setting the `EndpointId` (in some cases) in JavaScript.
- Changed parameter order in `AddIntent` in JavaScript, and added missing `AddIntent` JavaScript signature.

## Samples

- Added C++ and C# samples for pull and push stream usage in the [sample repository](#).

# Speech SDK 1.0.1

Reliability improvements and bug fixes:

- Fixed potential fatal error due to race condition in disposing recognizer
- Fixed potential fatal error in case of unset properties.
- Added additional error and parameter checking.
- Objective-C: Fixed possible fatal error caused by name overriding in NSString.
- Objective-C: Adjusted visibility of API
- JavaScript: Fixed regarding events and their payloads.
- Documentation improvements.

In our [sample repository](#), a new sample for JavaScript was added.

# Cognitive Services Speech SDK 1.0.0: 2018-September release

## New features

- Support for Objective-C on iOS. Check out our [Objective-C quickstart for iOS](#).
- Support for JavaScript in browser. Check out our [JavaScript quickstart](#).

## Breaking changes

- With this release, a number of breaking changes are introduced. Check [this page](#) for details.

# Cognitive Services Speech SDK 0.6.0: 2018-August release

## New features

- UWP apps built with the Speech SDK now can pass the Windows App Certification Kit (WACK). Check out the [UWP quickstart](#).
- Support for .NET Standard 2.0 on Linux (Ubuntu 16.04 x64).
- Experimental: Support Java 8 on Windows (64-bit) and Linux (Ubuntu 16.04 x64). Check out the [Java Runtime Environment quickstart](#).

### Functional change

- Expose additional error detail information on connection errors.

### Breaking changes

- On Java (Android), the `SpeechFactory.configureNativePlatformBindingWithDefaultCertificate` function no longer requires a path parameter. Now the path is automatically detected on all supported platforms.
- The get-accessor of the property `EndpointUrl` in Java and C# was removed.

### Bug fixes

- In Java, the audio synthesis result on the translation recognizer is implemented now.
- Fixed a bug that could cause inactive threads and an increased number of open and unused sockets.
- Fixed a problem, where a long-running recognition could terminate in the middle of the transmission.
- Fixed a race condition in recognizer shutdown.

## Cognitive Services Speech SDK 0.5.0: 2018-July release

### New features

- Support Android platform (API 23: Android 6.0 Marshmallow or higher). Check out the [Android quickstart](#).
- Support .NET Standard 2.0 on Windows. Check out the [.NET Core quickstart](#).
- Experimental: Support UWP on Windows (version 1709 or later).
  - Check out the [UWP quickstart](#).
  - Note: UWP apps built with the Speech SDK do not yet pass the Windows App Certification Kit (WACK).
- Support long-running recognition with automatic reconnection.

### Functional changes

- `StartContinuousRecognitionAsync()` supports long-running recognition.
- The recognition result contains more fields. They're offset from the audio beginning and duration (both in ticks) of the recognized text and additional values that represent recognition status, for example, `InitialSilenceTimeout` and `InitialBabbleTimeout`.
- Support `AuthorizationToken` for creating factory instances.

### Breaking changes

- Recognition events: `NoMatch` event type was merged into the `Error` event.
- `SpeechOutputFormat` in C# was renamed to `OutputFormat` to stay aligned with C++.
- The return type of some methods of the `AudioInputStream` interface changed slightly:
  - In Java, the `read` method now returns `long` instead of `int`.
  - In C#, the `Read` method now returns `uint` instead of `int`.
  - In C++, the `Read` and `GetFormat` methods now return `size_t` instead of `int`.
- C++: Instances of audio input streams now can be passed only as a `shared_ptr`.

### Bug fixes

- Fixed incorrect return values in the result when `RecognizeAsync()` times out.
- The dependency on media foundation libraries on Windows was removed. The SDK now uses Core Audio APIs.
- Documentation fix: Added a [regions](#) page to describe the supported regions.

#### **Known issue**

- The Speech SDK for Android doesn't report speech synthesis results for translation. This issue will be fixed in the next release.

## Cognitive Services Speech SDK 0.4.0: 2018-June release

#### **Functional changes**

- `AudioInputStream`

A recognizer now can consume a stream as the audio source. For more information, see the related [how-to guide](#).

- Detailed output format

When you create a `SpeechRecognizer`, you can request `Detailed` or `Simple` output format. The `DetailedSpeechRecognitionResult` contains a confidence score, recognized text, raw lexical form, normalized form, and normalized form with masked profanity.

#### **Breaking change**

- Changed to `SpeechRecognitionResult.Text` from `SpeechRecognitionResult.RecognizedText` in C#.

#### **Bug fixes**

- Fixed a possible callback issue in the USP layer during shutdown.
- If a recognizer consumed an audio input file, it was holding on to the file handle longer than necessary.
- Removed several deadlocks between the message pump and the recognizer.
- Fired a `NoMatch` result when the response from service is timed out.
- The media foundation libraries on Windows are delay loaded. This library is required for microphone input only.
- The upload speed for audio data is limited to about twice the original audio speed.
- On Windows, C# .NET assemblies now are strong named.
- Documentation fix: `Region` is required information to create a recognizer.

More samples have been added and are constantly being updated. For the latest set of samples, see the [Speech SDK samples GitHub repository](#).

## Cognitive Services Speech SDK 0.2.12733: 2018-May release

This release is the first public preview release of the Cognitive Services Speech SDK.

# About the Speech Devices SDK

12/4/2019 • 2 minutes to read • [Edit Online](#)

The [Speech service](#) works with a wide variety of devices and audio sources. Now, you can take your speech applications to the next level with matched hardware and software. The Speech Devices SDK is a pretuned library that's paired with purpose-built, microphone array development kits.

The Speech Devices SDK can help you:

- Rapidly test new voice scenarios.
- More easily integrate the cloud-based Speech service into your device.
- Create an exceptional user experience for your customers.

The Speech Devices SDK consumes the [Speech SDK](#). It uses the Speech SDK to send the audio that's processed by our advanced audio processing algorithm from the device's microphone array to the [Speech service](#). It uses multichannel audio to provide more accurate far-field [speech recognition](#) via noise suppression, echo cancellation, beamforming, and dereverberation.

You can also use the Speech Devices SDK to build ambient devices that have your own [customized keyword](#) so the cue that initiates a user interaction is unique to your brand.

The Speech Devices SDK facilitates a variety of voice-enabled scenarios, such as [voice assistants](#), drive-thru ordering systems, [conversation transcription](#), and smart speakers. You can respond to users with text, speak back to them in a default or [custom voice](#), provide search results, [translate](#) to other languages, and more. We look forward to seeing what you build!

## Get the Speech Devices SDK

### Android

For Android devices download the latest version of the [Android Speech Devices SDK](#).

### Windows

For Windows the sample application is provided as a cross-platform Java application. Download the latest version of the [JRE Speech Devices SDK](#). The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Windows. It runs on a 64-bit Java 8 runtime environment (JRE).

### Linux

For Linux the sample application is provided as a cross-platform Java application. Download the latest version of the [JRE Speech Devices SDK](#). The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Linux (Ubuntu 16.04, Ubuntu 18.04, Debian 9). It runs on a 64-bit Java 8 runtime environment (JRE).

## Next steps

[Choose your speech device](#)

[Get a Speech service subscription key for free](#)

# Get the Cognitive Services Speech Devices SDK

12/4/2019 • 2 minutes to read • [Edit Online](#)

The Speech Devices SDK is a pretuned library designed to work with purpose-built development kits, and varying microphone array configurations.

## Choose a Development kit

| DEVICES                                                                                                                                           | SPECIFICATION                                                         | DESCRIPTION                                                                                                                                                                                                                                                                                                                                               | SCENARIOS                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <a href="#">Roobo Smart Audio Dev Kit Setup / Quickstart</a><br> | 7 Mic Array, ARM SOC, WIFI, Audio Out, IO.<br><a href="#">Android</a> | The first Speech Devices SDK to adapt Microsoft Mic Array and front processing SDK, for developing high-quality transcription and speech scenarios                                                                                                                                                                                                        | Conversation Transcription, Smart Speaker, Voice Agent, Wearable                          |
| <a href="#">Azure Kinect DK Setup / Quickstart</a><br>         | 7 Mic Array RGB and Depth cameras.<br><a href="#">Windows/Linux</a>   | A developer kit with advanced artificial intelligence (AI) sensors for building sophisticated computer vision and speech models. It combines a best-in-class spatial microphone array and depth camera with a video camera and orientation sensor—all in one small device with multiple modes, options, and SDKs to accommodate a range of compute types. | Conversation Transcription, Robotics, Smart Building                                      |
| <a href="#">Roobo Smart Audio Dev Kit 2</a><br>                | 7 Mic Array, ARM SOC, WIFI, Bluetooth, IO.<br>Linux                   | The 2nd generation Speech Devices SDK that provides alternative OS and more features in a cost effective reference design.                                                                                                                                                                                                                                | Conversation Transcription, Smart Speaker, Voice Agent, Wearable                          |
| <a href="#">URbetter T11 Development Board</a><br>             | 7 Mic Array, ARM SOC, WIFI, Ethernet, HDMI, USB Camera.<br>Linux      | An industry level Speech Devices SDK that adapts Microsoft Mic array and supports extended I/O such as HDMI/Ethernet and more USB peripherals                                                                                                                                                                                                             | Conversation Transcription, Education, Hospital, Robots, OTT Box, Voice Agent, Drive Thru |

# Download the Speech Devices SDK

Download the [Speech Devices SDK](#).

## Next steps

[Get started with the Speech Devices SDK](#)

# Speech Devices SDK Microphone array recommendations

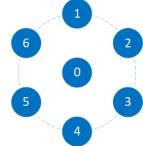
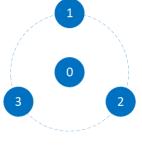
11/14/2019 • 4 minutes to read • [Edit Online](#)

In this article, you learn how to design a microphone array for the Speech Devices SDK.

The Speech Devices SDK works best with a microphone array that has been designed according to the following guidelines, including the microphone geometry and component selection. Guidance is also given on integration and electrical considerations.

## Microphone geometry

The following array geometries are recommended for use with the Microsoft Audio Stack. Location of sound sources and rejection of ambient noise is improved with greater number of microphones with dependencies on specific applications, user scenarios, and the device form factor.

|          | CIRCULAR ARRAY                                                                     | LINEAR ARRAY                                                                       |                                     |
|----------|------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|-------------------------------------|
|          |  |  |                                     |
| # Mics   | 7                                                                                  | 4                                                                                  | 4                                   |
| Geometry | 6 Outer, 1 Center,<br>Radius = 42.5 mm,<br>Evenly Spaced                           | 3 Outer, 1 Center,<br>Radius = 42.5 mm,<br>Evenly Spaced                           | Length = 120 mm,<br>Spacing = 40 mm |

Microphone channels should be ordered according to the numbering depicted for each above array, increasing from 0. The Microsoft Audio Stack will require an additional reference stream of audio playback to perform echo cancellation.

## Component selection

Microphone components should be selected to accurately reproduce a signal free of noise and distortion.

The recommended properties when selecting microphones are:

| PARAMETER          | RECOMMENDED                                         |
|--------------------|-----------------------------------------------------|
| SNR                | >= 65 dB (1 kHz signal 94 dB SPL, A-weighted noise) |
| Amplitude Matching | ± 1 dB @ 1 kHz                                      |
| Phase Matching     | ± 2° @ 1 kHz                                        |

| PARAMETER                     | RECOMMENDED                                                                          |
|-------------------------------|--------------------------------------------------------------------------------------|
| Acoustic Overload Point (AOP) | >= 120 dB SPL (THD = 10%)                                                            |
| Bit Rate                      | Minimum 24-bit                                                                       |
| Sampling Rate                 | Minimum 16 kHz*                                                                      |
| Frequency Response            | ± 3 dB, 200-8000 Hz Floating Mask*                                                   |
| Reliability                   | Storage Temperature Range -40°C to 70°C<br>Operating Temperature Range -20°C to 55°C |

\*Higher sampling rates or "wider" frequency ranges may be necessary for high-quality communications (VoIP) applications

Good component selection must be paired with good electroacoustic integration in order to avoid impairing the performance of the components used. Unique use cases may also necessitate additional requirements (for example: operating temperature ranges).

## Microphone array integration

The performance of the microphone array when integrated into a device will differ from the component specification. It is important to ensure that the microphones are well matched after integration. Therefore the device performance measured after any fixed gain or EQ should meet the following recommendations:

| PARAMETER          | RECOMMENDED                                        |
|--------------------|----------------------------------------------------|
| SNR                | > 63 dB (1 kHz signal 94 dB SPL, A-weighted noise) |
| Output Sensitivity | -26 dBFS/Pa @ 1 kHz (recommended)                  |
| Amplitude Matching | ± 2 dB, 200-8000 Hz                                |
| THD%*              | ≤ 1%, 200-8000 Hz, 94 dB SPL, 5th Order            |
| Frequency Response | ± 6 dB, 200-8000 Hz Floating Mask**                |

\*\*A low distortion speaker is required to measure THD (e.g. Neumann KH120)

\*\*"Wider" frequency ranges may be necessary for high-quality communications (VoIP) applications

## Speaker integration recommendations

As echo cancellation is necessary for speech recognition devices that contain speakers, additional recommendations are provided for speaker selection and integration.

| PARAMETER                | RECOMMENDED                                                                                                        |
|--------------------------|--------------------------------------------------------------------------------------------------------------------|
| Linearity Considerations | No non-linear processing after speaker reference, otherwise a hardware-based loopback reference stream is required |
| Speaker Loopback         | Provided via WASAPI, private APIs, custom ALSA plug-in (Linux), or provided via firmware channel                   |

| PARAMETER                    | RECOMMENDED                                                                                                                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| THD%                         | 3rd Octave Bands minimum 5th Order, 70 dBA Playback @ 0.8 m ≤ 6.3%, 315-500 Hz ≤ 5%, 630-5000 Hz                                                                                                  |
| Echo Coupling to Microphones | > -10 dB TCLw using ITU-T G.122 Annex B.4 method, normalized to mic level<br>TCLw = TCLwmeasured + (Measured Level - Target Output Sensitivity)<br>TCLw = TCLwmeasured + (Measured Level - (-26)) |

## Integration design architecture

The following guidelines for architecture are necessary when integrating microphones into a device:

| PARAMETER           | RECOMMENDATION                                                                                                                                                       |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Mic Port Similarity | All microphone ports are same length in array                                                                                                                        |
| Mic Port Dimensions | Port size Ø0.8-1.0 mm. Port Length / Port Diameter < 2                                                                                                               |
| Mic Sealing         | Sealing gaskets uniformly implemented in stack-up.<br>Recommend > 70% compression ratio for foam gaskets                                                             |
| Mic Reliability     | Mesh should be used to prevent dust and ingress (between PCB for bottom ported microphones and sealing gasket/top cover)                                             |
| Mic Isolation       | Rubber gaskets and vibration decoupling through structure, particularly for isolating any vibration paths due to integrated speakers                                 |
| Sampling Clock      | Device audio must be free of jitter and drop-outs with low drift                                                                                                     |
| Record Capability   | The device must be able to record individual channel raw streams simultaneously                                                                                      |
| USB                 | All USB audio input devices must set descriptors according to the <a href="#">USB Audio Devices Rev3 Spec</a>                                                        |
| Microphone Geometry | Drivers must implement <a href="#">Microphone Array Geometry Descriptors</a> correctly                                                                               |
| Discoverability     | Devices must not have any undiscoverable or uncontrollable hardware, firmware, or 3rd party software-based non-linear audio processing algorithms to/from the device |
| Capture Format      | Capture formats must use a minimum sampling rate of 16 kHz and recommended 24-bit depth                                                                              |

## Electrical architecture considerations

Where applicable, arrays may be connected to a USB host (such as an SoC that runs the Microsoft Audio Stack) and interfaces to speech services or other applications.

Hardware components such as PDM-to-TDM conversion should ensure that the dynamic range and SNR of the microphones is preserved within re-samplers.

High-speed USB Audio Class 2.0 should be supported within any audio MCUs in order to provide the necessary bandwidth for up to seven channels at higher sample rates and bit depths.

## Next steps

[Learn more about the Speech devices SDK](#)

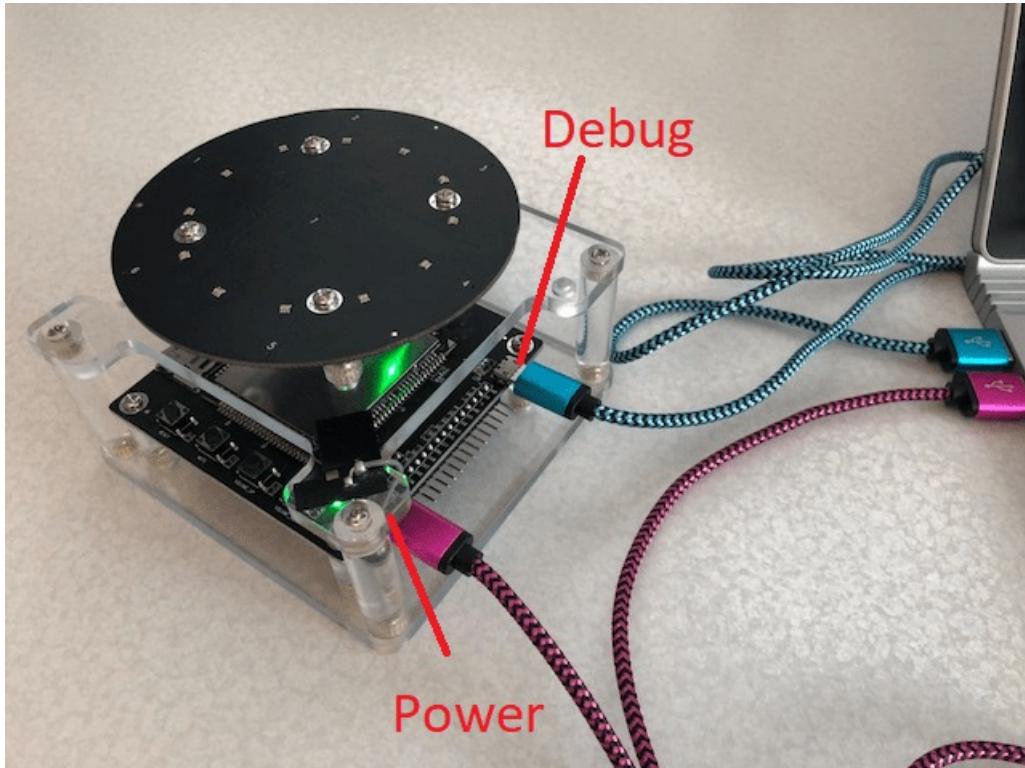
# Device: Roobo Smart Audio Dev Kit

12/4/2019 • 2 minutes to read • [Edit Online](#)

This article provides device specific information for the Roobo Smart Audio Dev Kit.

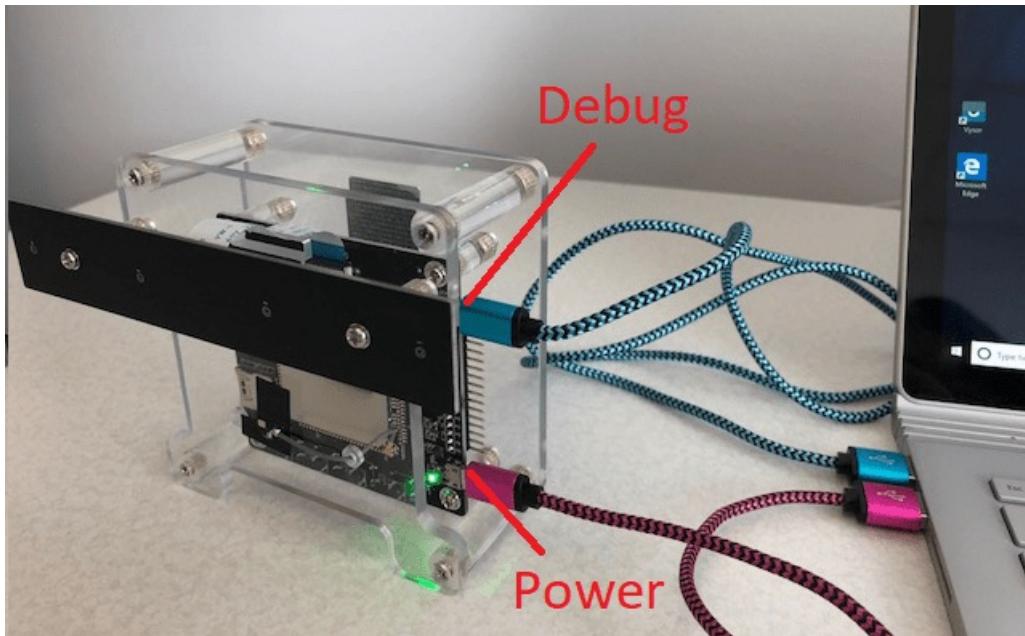
## Set up the development kit

1. The development kit has two micro USB connectors. The left connector is to power the development kit and is highlighted as Power in the image below. The right one is to control it, and is marked Debug in the image.



2. Power the development kit by using a micro USB cable to connect the power port to a PC or power adapter. A green power indicator will light up under the top board.
3. To control the development kit, connect the debug port to a computer by using a second micro USB cable. It is essential to use a high quality cable to ensure reliable communications.
4. Orient your development kit for either the circular or linear configuration.

| DEVELOPMENT KIT CONFIGURATION | ORIENTATION                                                             |
|-------------------------------|-------------------------------------------------------------------------|
| Circular                      | Upright, with microphones facing the ceiling                            |
| Linear                        | On its side, with microphones facing you (shown in the following image) |



5. Install the certificates and set the permissions of the sound device. Type the following commands in a Command Prompt window:

```
adb push C:\SDSDK\Android-Sample-Release\scripts\roobo_setup.sh /data/
adb shell
cd /data/
chmod 777 roobo_setup.sh
../roobo_setup.sh
exit
```

#### NOTE

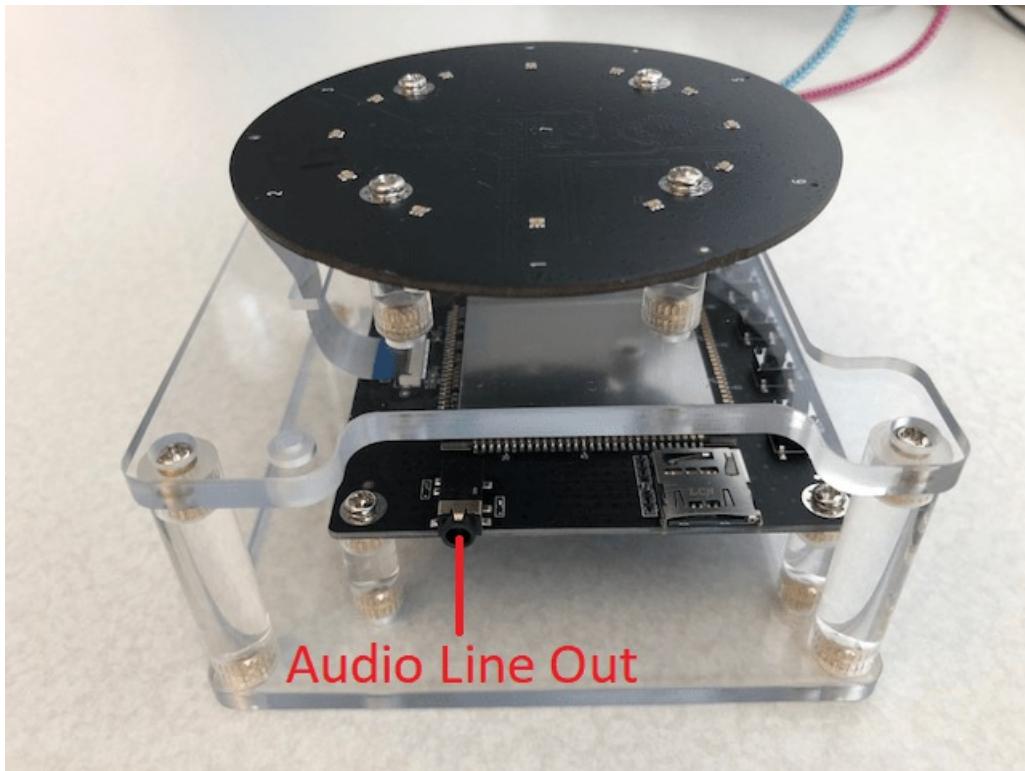
These commands use the Android Debug Bridge, `adb.exe`, which is part of the Android Studio installation. This tool is located in `C:\Users[user name]\AppData\Local\Android\Sdk\platform-tools`. You can add this directory to your path to make it more convenient to invoke `adb`. Otherwise, you must specify the full path to your installation of `adb.exe` in every command that invokes `adb`.

If you see an error `no devices/emulators found` then check your USB cable is connected and is a high quality cable. You can use `adb devices` to check that your computer can talk to the development kit as it will return a list of devices.

#### TIP

Mute your PC's microphone and speaker to be sure you are working with the development kit's microphones. This way, you won't accidentally trigger the device with audio from the PC.

6. If you want to attach a speaker to the dev kit, you can connect it to the audio line out. You should choose a good-quality speaker with a 3.5mm analog plug.



## Development information

For more development information, see the [Roobo development guide](#).

## Audio

Roobo provides a tool that captures all audio to flash memory. It might help you troubleshoot audio issues. A version of the tool is provided for each development kit configuration. On the [Roobo site](#), select your device, and then select the **Roobo Tools** link at the bottom of the page.

## Next steps

- [Run the Android sample app](#)

# Quickstart: Run the Speech Devices SDK sample app on Windows

12/4/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to use the Speech Devices SDK for Windows to build a speech-enabled product or use it as a [Conversation Transcription](#) device. For Conversation Transcription only the [Azure Kinect DK](#) is supported. For other speech use linear mic arrays that provide a microphone array geometry are supported.

The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Windows. It runs on a 64-bit Java 8 runtime environment (JRE).

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

The source code for the [sample application](#) is included with the Speech Devices SDK. It's also [available on GitHub](#).

## Prerequisites

This quickstart requires:

- Operating System: 64-bit Windows
- A microphone array such as [Azure Kinect DK](#)
- [Eclipse Java IDE](#)
- [Java 8 or JDK 8](#) only.
- [Microsoft Visual C++ Redistributable](#)
- An Azure subscription key for the Speech service. [Get one for free](#).
- Download the latest version of the [Speech Devices SDK](#) for Java, and extract the .zip to your working directory.

### NOTE

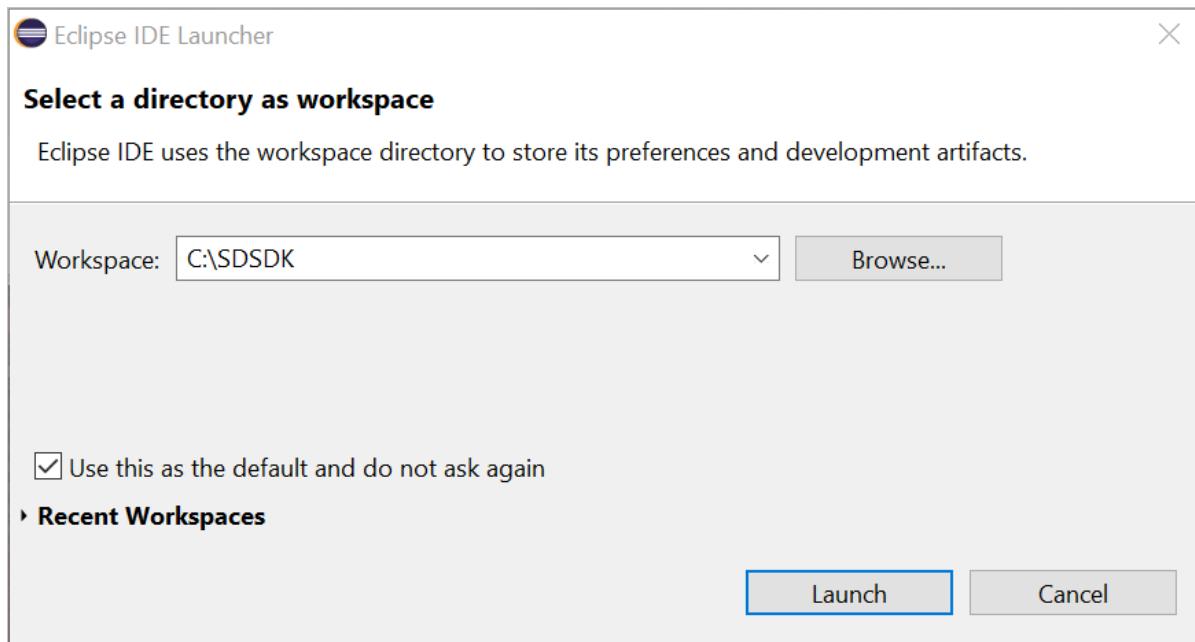
The JRE-Sample-Release.zip file includes the JRE sample app and this quickstart assumes that the app is extracted to C:\SDSDK\JRE-Sample-Release

Conversation Transcription is currently only available for "en-US" and "zh-CN", in the "centralus" and "eastasia" regions. You must have a speech key in one of those regions to use Conversation Transcription.

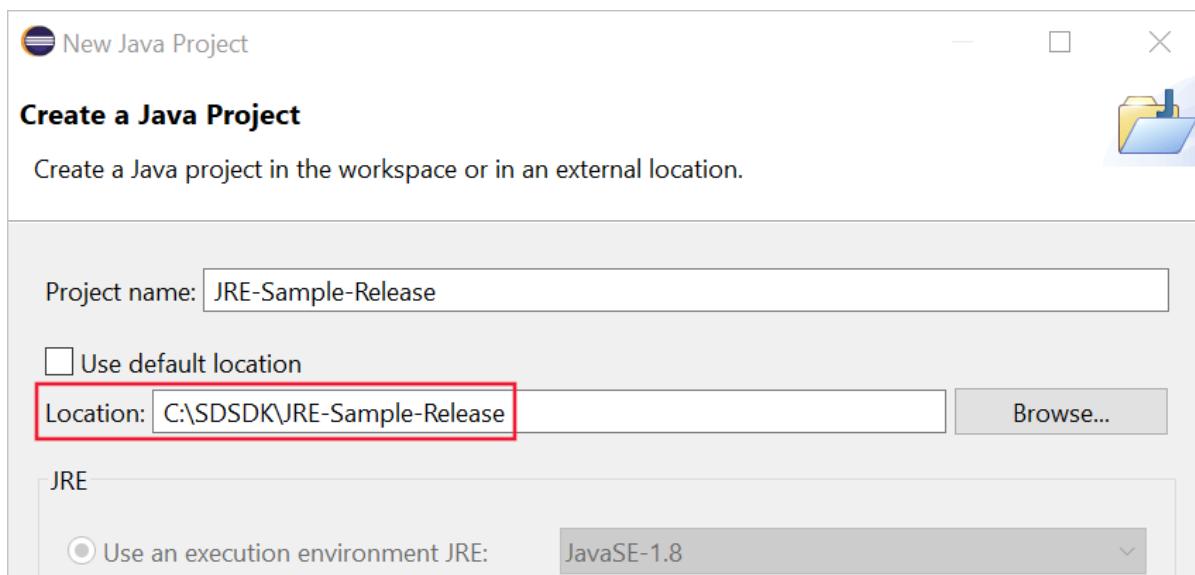
If you plan to use the intents you'll need a [Language Understanding Service \(LUIS\)](#) subscription. To learn more about LUIS and intent recognition, see [Recognize speech intents with LUIS, C#](#). A [sample LUIS model](#) is available for this app.

## Create and configure the project

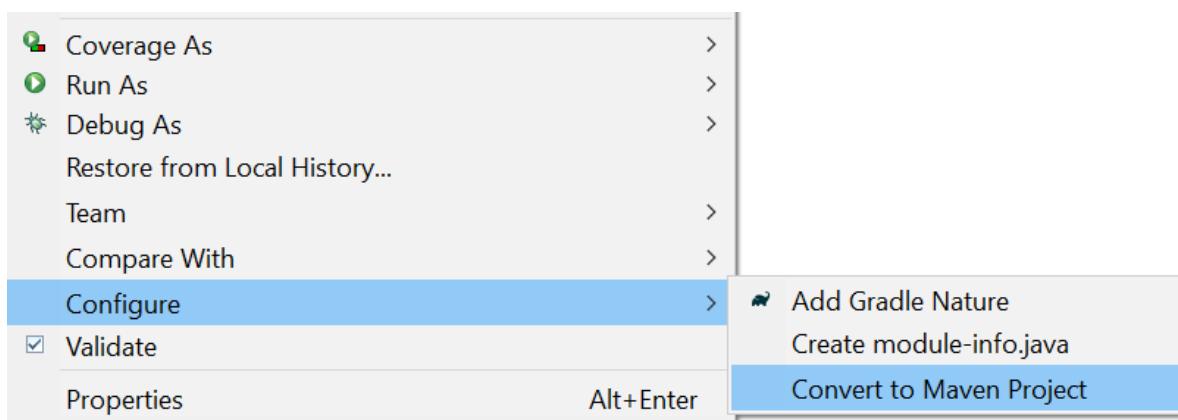
1. Start Eclipse.
2. In the **Eclipse IDE Launcher**, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.



3. In a moment, the main window of the Eclipse IDE appears. Close the Welcome screen if one is present.
4. From the Eclipse menu bar, create a new project by choosing **File > New > Java Project**. If not available choose **Project** and then **Java Project**.
5. The **New Java Project** wizard starts. **Browse** for the location of the sample project. Select **Finish**.



6. In the **Package explorer**, right-click your project. Choose **Configure > Convert to Maven Project** from the context menu. Select **Finish**.



7. Open the pom.xml file and edit it.

At the end of the file, before the closing tag `</project>`, create `repositories` and `dependencies` elements, as shown here, and ensure the `version` matches your current version:

```
<repositories>
 <repository>
 <id>maven-cognitiveservices-speech</id>
 <name>Microsoft Cognitive Services Speech Maven Repository</name>
 <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
 </repository>
</repositories>

<dependencies>
 <dependency>
 <groupId>com.microsoft.cognitiveservices.speech</groupId>
 <artifactId>client-sdk</artifactId>
 <version>1.7.0</version>
 </dependency>
</dependencies>
```

8. Copy the contents of **Windows-x64** to the Java Project location, eg `C:\SDSDK\JRE-Sample-Release`
9. Copy `kws.table`, `participants.properties` and `Microsoft.CognitiveServices.Speech.extension.pma.dll` into the project folder **target\classes**

## Configure the sample application

1. Add your speech subscription key to the source code. If you want to try intent recognition, also add your [Language Understanding service](#) subscription key and application ID.

For speech and LUIS, your information goes into `FunctionsList.java`:

```
// Subscription
private static String SpeechSubscriptionKey = "<enter your subscription info here>";
private static String SpeechRegion = "westus"; // You can change this if your speech region is
different.
private static String LuisSubscriptionKey = "<enter your subscription info here>";
private static String LuisRegion = "westus2"; // you can change this, if you want to test the intent,
and your LUIS region is different.
private static String LuisAppId = "<enter your LUIS AppId>";
```

If you are using conversation transcription, your speech key and region information are also needed in `Cts.java`:

```
private static final String CTSKey = "<Conversation Transcription Service Key>";
private static final String CTSRegion="<Conversation Transcription Service Region>";// Region may be
"centralus" or "eastasia"
```

2. The default keyword (keyword) is "Computer". You can also try one of the other provided keywords, like "Machine" or "Assistant". The resource files for these alternate keywords are in the Speech Devices SDK, in the keyword folder. For example, `C:\SDSDK\JRE-Sample-Release\keyword\Computer` contains the files used for the keyword "Computer".

### TIP

You can also [create a custom keyword](#).

To use a new keyword, update the following line in `FunctionsList.java`, and copy the keyword to your app.

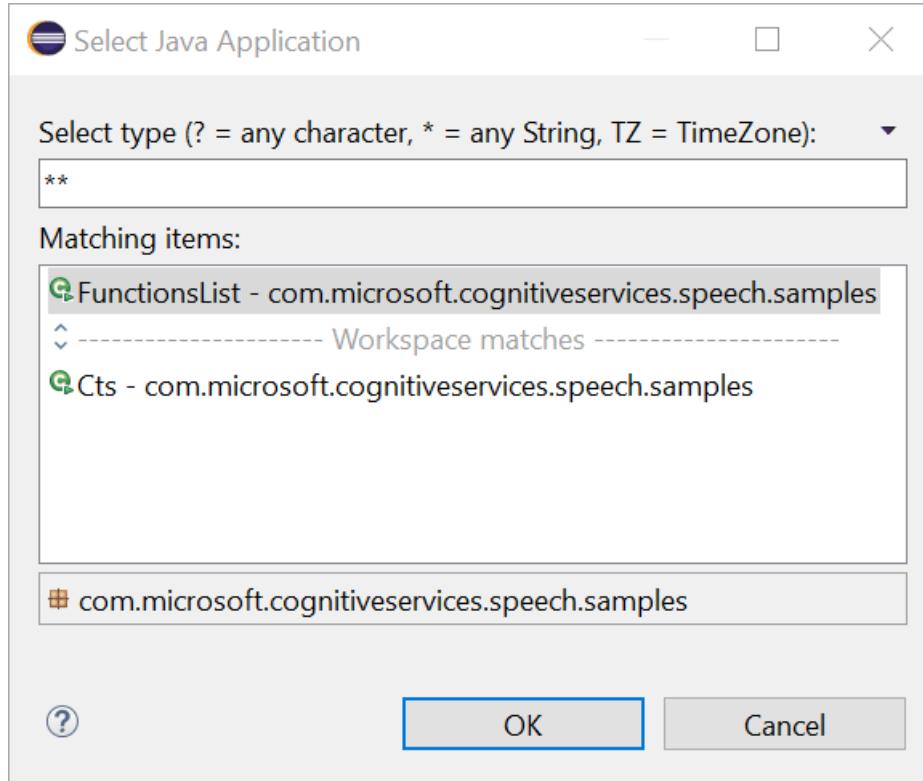
For example, to use the keyword 'Machine' from the keyword package `machine.zip`:

- Copy the `kws.table` file from the zip package into the project folder **target/classes**.
- Update the `FunctionsList.java` with the keyword name:

```
private static final String Keyword = "Machine";
```

## Run the sample application from Eclipse

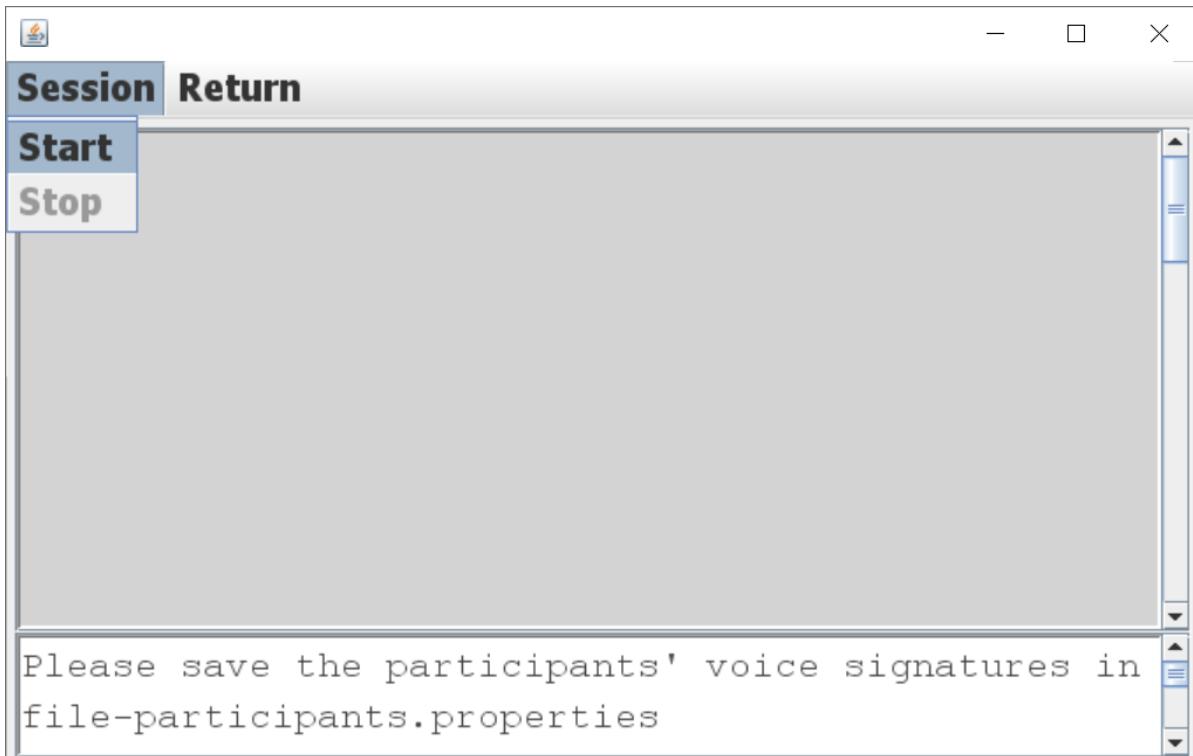
1. From the Eclipse menu bar, **Run > Run As > Java Application**. Then select **FunctionsList** and **OK**.



2. The Speech Devices SDK example application starts and displays the following options:

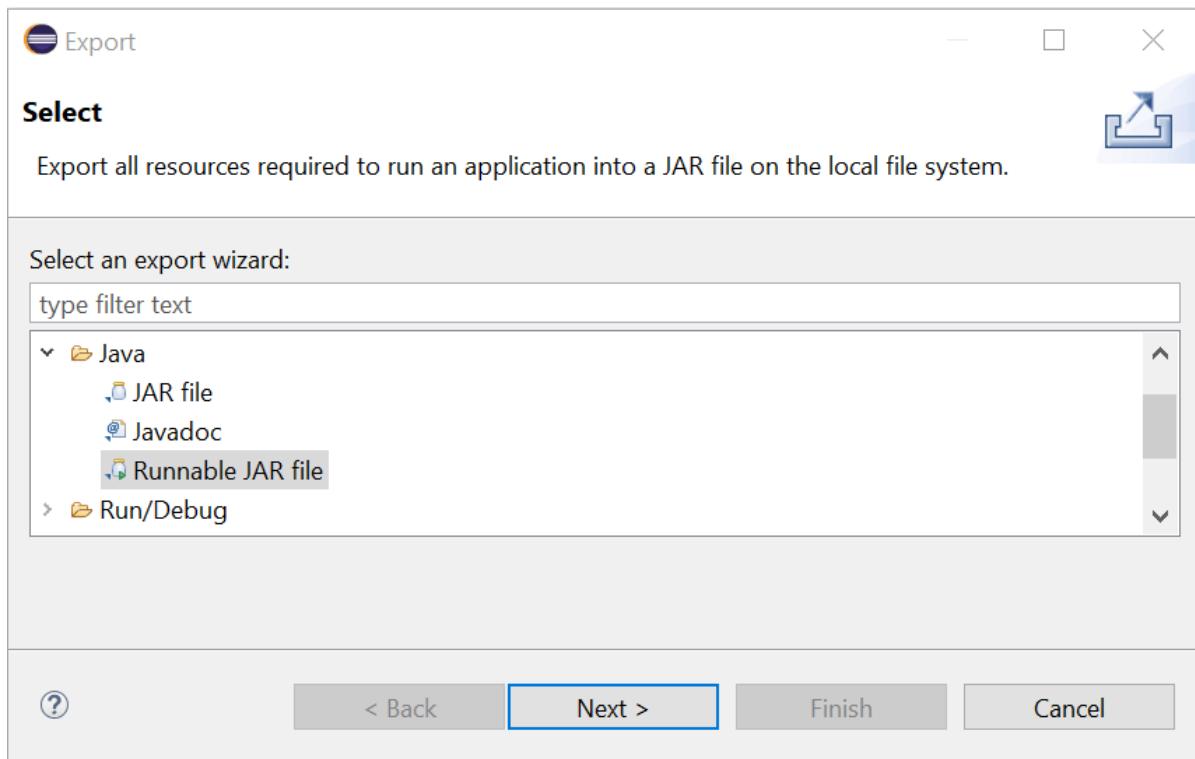


3. Try the new **Conversation Transcription** demo. Start transcribing with **Session > Start**. By default everyone is a guest. However, if you have participant's voice signatures they can be put into a file `participants.properties` in the project folder **target/classes**. To generate the voice signature, look at [Transcribe conversations \(SDK\)](#).

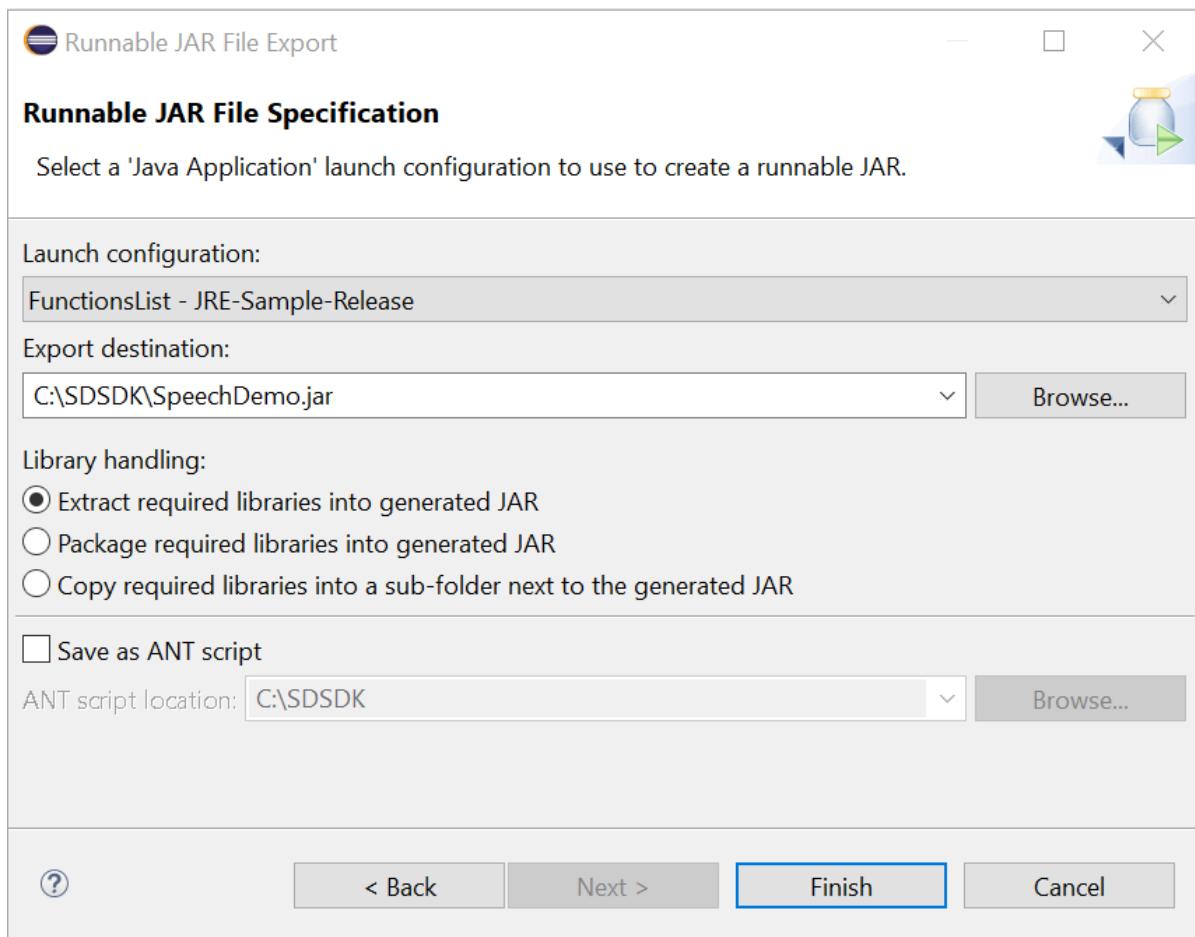


## Create and run a standalone application

1. In the **Package explorer**, right-click your project. Choose **Export**.
2. The **Export** window appears. Expand **Java** and select **Runnable JAR file** and then select **Next**.



3. The **Runnable JAR File Export** window appears. Choose an **Export destination** for the application, and then select **Finish**.



4. Please put `kws.table`, `participants.properties`, `unimic_runtime.dll`, `pma.dll` and `Microsoft.CognitiveServices.Speech.extension.pma.dll` in the destination folder chosen above as these files are needed by the application.
5. To run the standalone application

```
java -jar SpeechDemo.jar
```

## Next steps

[Review the release notes](#)

# Quickstart: Run the Speech Devices SDK sample app on Linux

12/4/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to use the Speech Devices SDK for Linux to build a speech-enabled product or use it as a [Conversation Transcription](#) device. Currently only the [Azure Kinect DK](#) is supported.

The application is built with the Speech SDK package, and the Eclipse Java IDE (v4) on 64-bit Linux (Ubuntu 16.04, Ubuntu 18.04, Debian 9). It runs on a 64-bit Java 8 runtime environment (JRE).

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

The source code for the [sample application](#) is included with the Speech Devices SDK. It's also [available on GitHub](#).

## Prerequisites

This quickstart requires:

- Operating System: 64-bit Linux (Ubuntu 16.04, Ubuntu 18.04, Debian 9)
- [Azure Kinect DK](#)
- [Eclipse Java IDE](#)
- [Java 8 or JDK 8](#) only.
- An Azure subscription key for the Speech service. [Get one for free](#).
- Download the latest version of the [Speech Devices SDK](#) for Java, and extract the .zip to your working directory.

### NOTE

The JRE-Sample-Release.zip file includes the JRE sample app and this quickstart assumes that the app is extracted to /home/wcaltest/JRE-Sample-Release

Make sure these dependencies are installed before starting Eclipse.

- On Ubuntu:

```
sudo apt-get update
sudo apt-get install libssl1.0.0 libasound2
```

- On Debian 9:

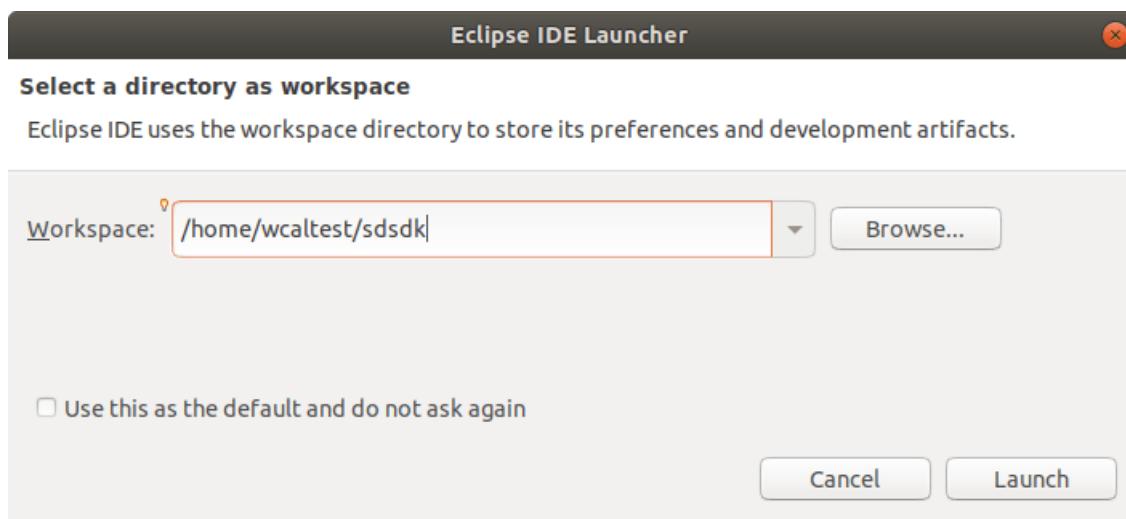
```
sudo apt-get update
sudo apt-get install libssl1.0.2 libasound2
```

Conversation Transcription is currently only available for "en-US" and "zh-CN", in the "centralus" and "eastasia" regions. You must have a speech key in one of those regions to use Conversation Transcription.

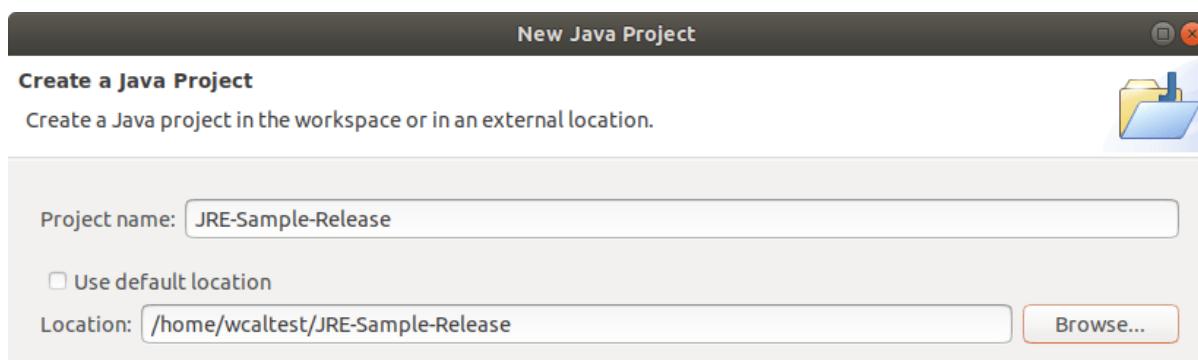
If you plan to use the intents you'll need a [Language Understanding Service \(LUIS\)](#) subscription. To learn more about LUIS and intent recognition, see [Recognize speech intents with LUIS, C#](#). A [sample LUIS model](#) is available for this app.

## Create and configure the project

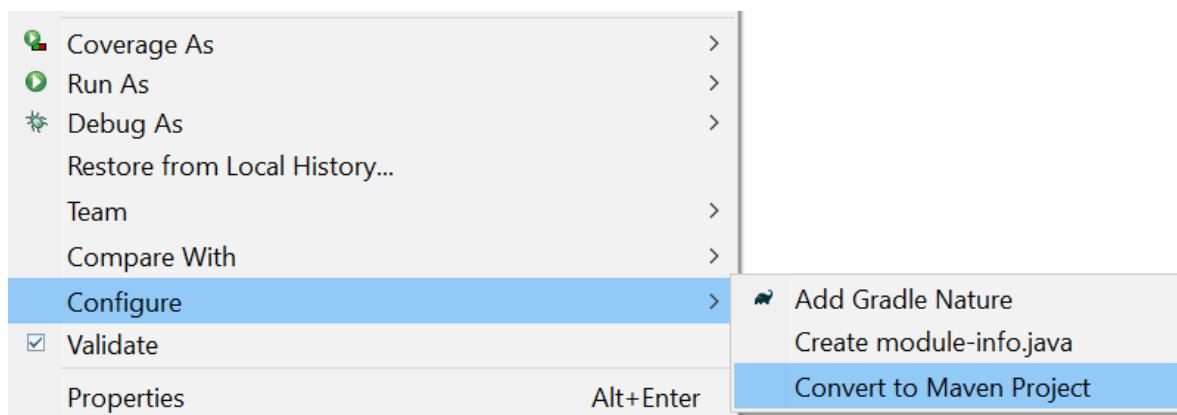
1. Start Eclipse.
2. In the **Eclipse IDE Launcher**, in the **Workspace** field, enter the name of a new workspace directory. Then select **Launch**.



3. In a moment, the main window of the Eclipse IDE appears. Close the Welcome screen if one is present.
4. From the Eclipse menu bar, create a new project by choosing **File > New > Java Project**. If not available choose **Project** and then **Java Project**.
5. The **New Java Project** wizard starts. **Browse** for the location of the sample project. Select **Finish**.



6. In the **Package explorer**, right-click your project. Choose **Configure > Convert to Maven Project** from the context menu. Select **Finish**.



7. Open the pom.xml file and edit it.

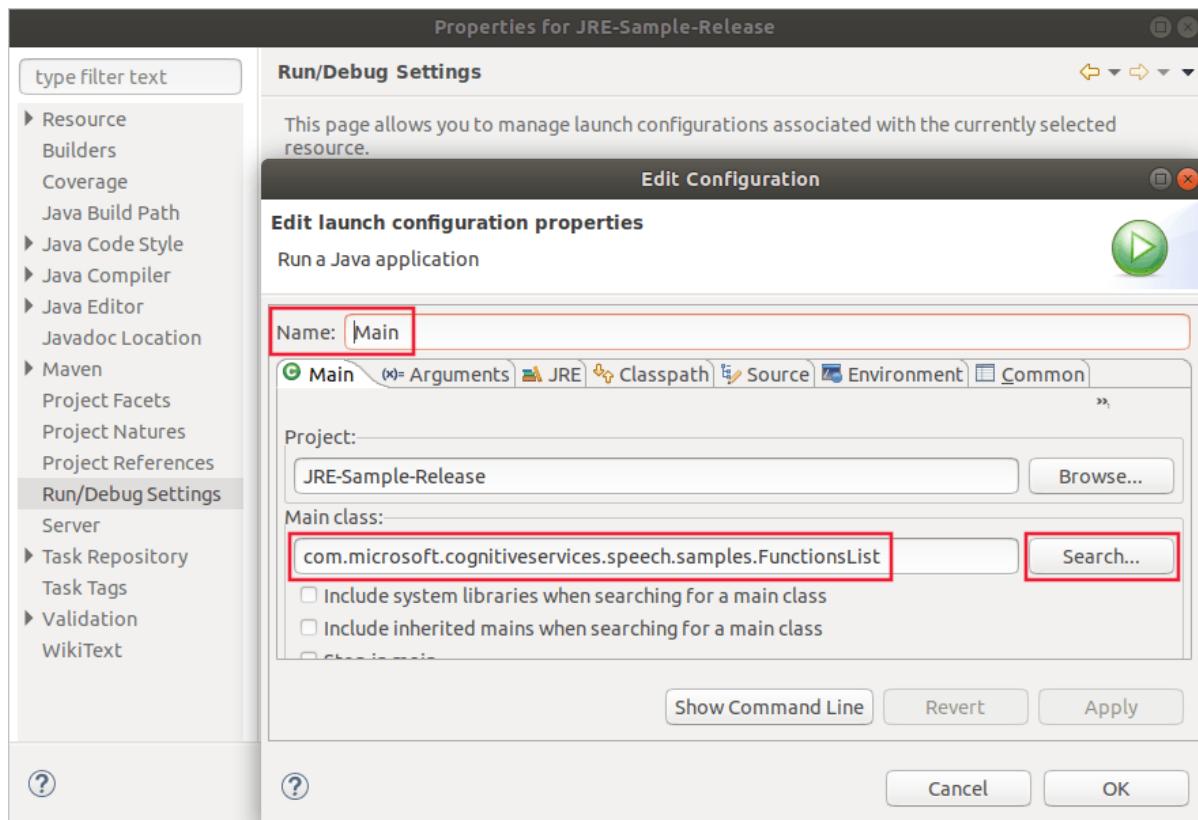
At the end of the file, before the closing tag `</project>`, create `repositories` and `dependencies` elements,

as shown here, and ensure the `version` matches your current version:

```
<repositories>
 <repository>
 <id>maven-cognitiveservices-speech</id>
 <name>Microsoft Cognitive Services Speech Maven Repository</name>
 <url>https://csspeechstorage.blob.core.windows.net/maven/</url>
 </repository>
</repositories>

<dependencies>
 <dependency>
 <groupId>com.microsoft.cognitiveservices.speech</groupId>
 <artifactId>client-sdk</artifactId>
 <version>1.7.0</version>
 </dependency>
</dependencies>
```

8. In the **Package explorer**, right-click your project. Choose **Properties**, then **Run/Debug Settings > New... > Java Application**.
9. The **Edit Configuration** window appears. In the **Name** field enter **Main**, and use **Search** for the **Main Class** to find and select **com.microsoft.cognitiveservices.speech.samples.FunctionsList**.



10. Copy the audio binaries for your target architecture, from either **Linux-arm** or **Linux-x64**, to the Java Project location, eg **/home/wcaltest/JRE-Sample-Release**
11. Also from the **Edit Configuration** window select the **Environment** page and **New**. The **New Environment Variable** window appears. In the **Name** field enter **LD\_LIBRARY\_PATH** and in the **value** field enter the folder containing the \*.so files, for example **/home/wcaltest/JRE-Sample-Release**
12. Copy `kws.table` and `participants.properties` into the project folder **target/classes**

## Configure the sample application

1. Add your speech subscription key to the source code. If you want to try intent recognition, also add your

Language Understanding service subscription key and application ID.

For speech and LUIS, your information goes into `FunctionsList.java`:

```
// Subscription
private static String SpeechSubscriptionKey = "<enter your subscription info here>";
private static String SpeechRegion = "westus"; // You can change this if your speech region is
different.
private static String LuisSubscriptionKey = "<enter your subscription info here>";
private static String LuisRegion = "westus2"; // you can change this, if you want to test the intent,
and your LUIS region is different.
private static String LuisAppId = "<enter your LUIS AppId>";
```

If you are using conversation transcription, your speech key and region information are also needed in

`Cts.java`:

```
private static final String CTSKey = "<Conversation Transcription Service Key>";
private static final String CTSRegion="<Conversation Transcription Service Region>";// Region may be
"centralus" or "eastasia"
```

2. The default keyword (keyword) is "Computer". You can also try one of the other provided keywords, like "Machine" or "Assistant". The resource files for these alternate keywords are in the Speech Devices SDK, in the keyword folder. For example, `/home/wcaltest/JRE-Sample-Release/keyword/Computer` contains the files used for the keyword "Computer".

**TIP**

You can also [create a custom keyword](#).

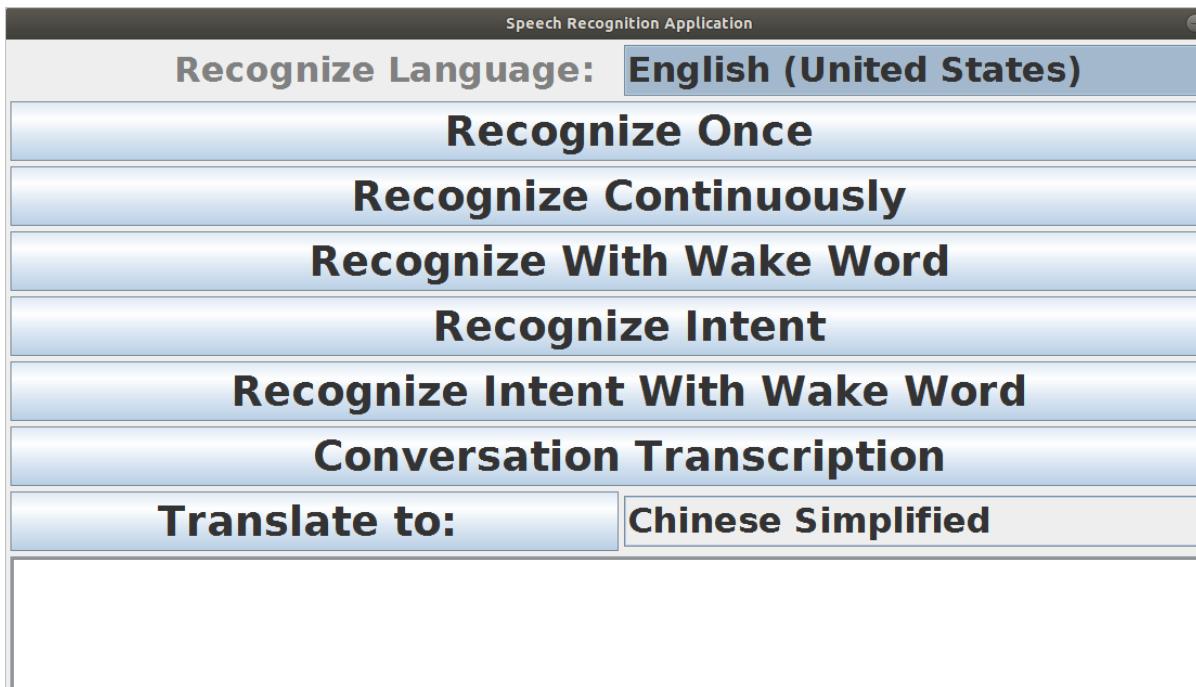
To use a new keyword, update the following line in `FunctionsList.java`, and copy the keyword to your app. For example, to use the keyword 'Machine' from the keyword package `machine.zip`:

- Copy the `kws.table` file from the zip package into the project folder **target/classes**.
- Update the `FunctionsList.java` with the keyword name:

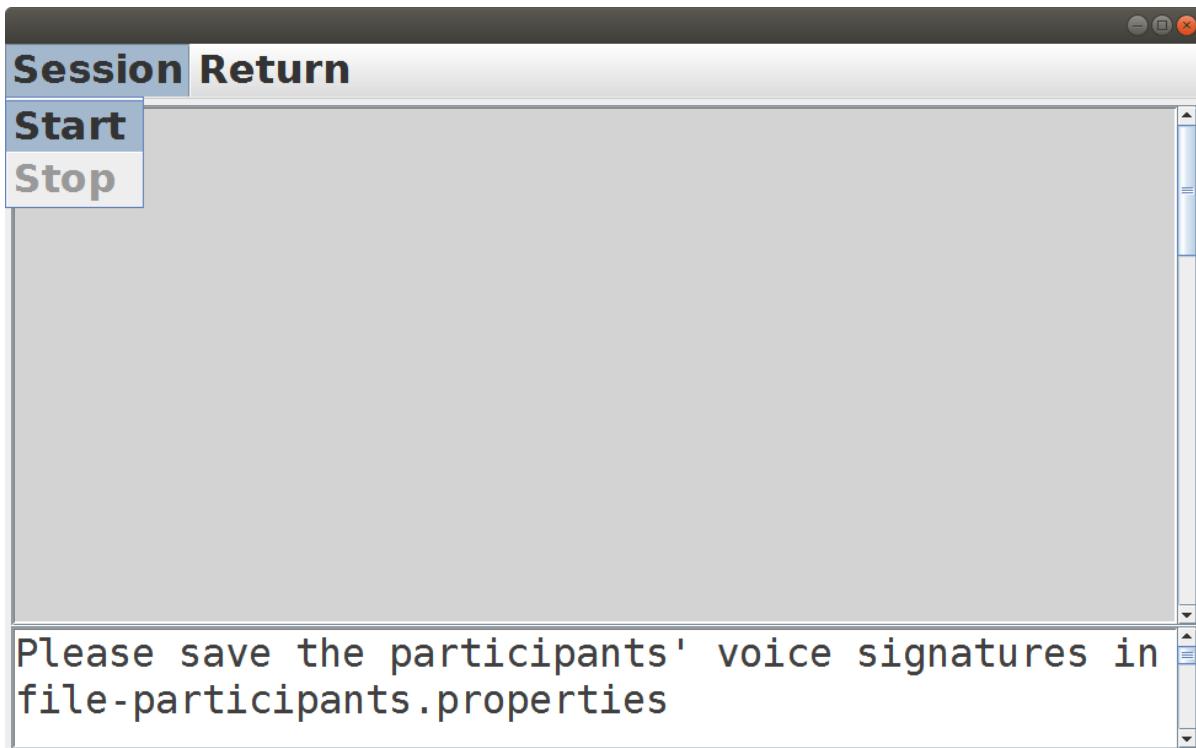
```
private static final String Keyword = "Machine";
```

## Run the sample application from Eclipse

1. From the Eclipse menu bar, **Run > Run**
2. The Speech Devices SDK example application starts and displays the following options:

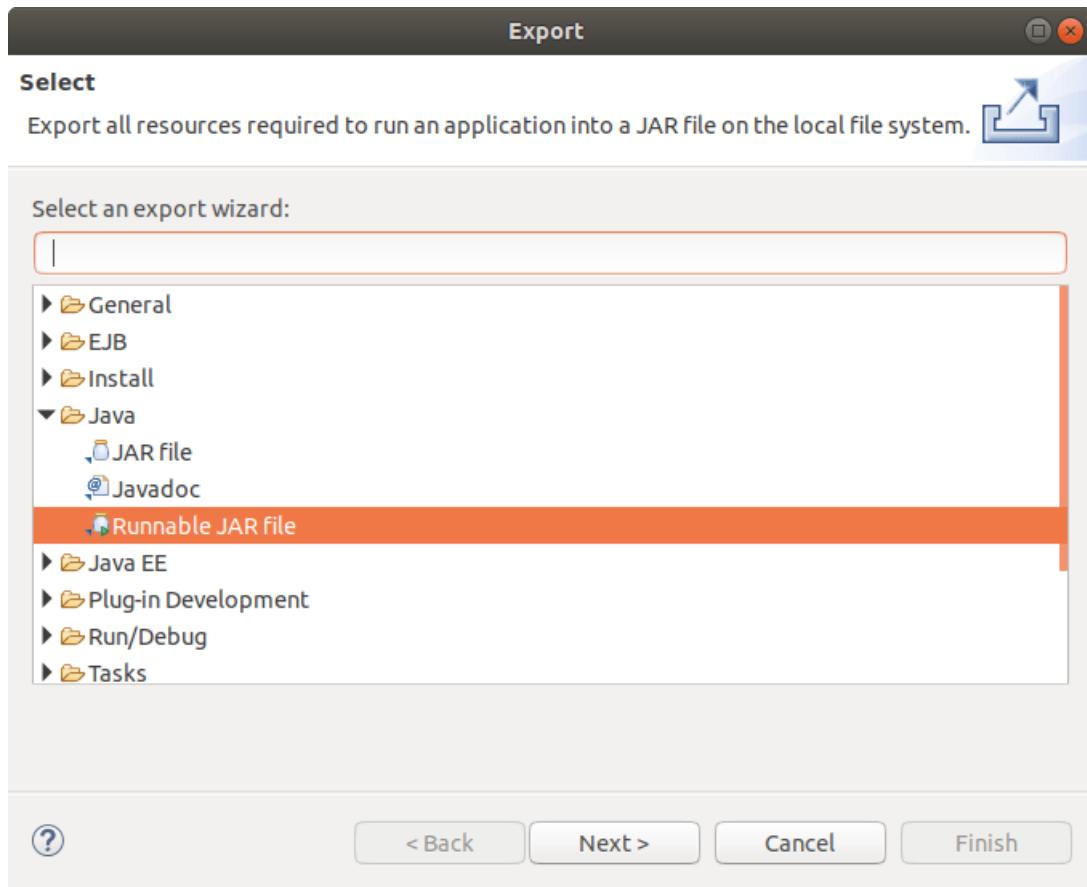


3. Try the new **Conversation Transcription** demo. Start transcribing with **Session > Start**. By default everyone is a guest. However, if you have participant's voice signatures they can be put into `participants.properties` in the project folder **target/classes**. To generate the voice signature, look at [Transcribe conversations \(SDK\)](#).

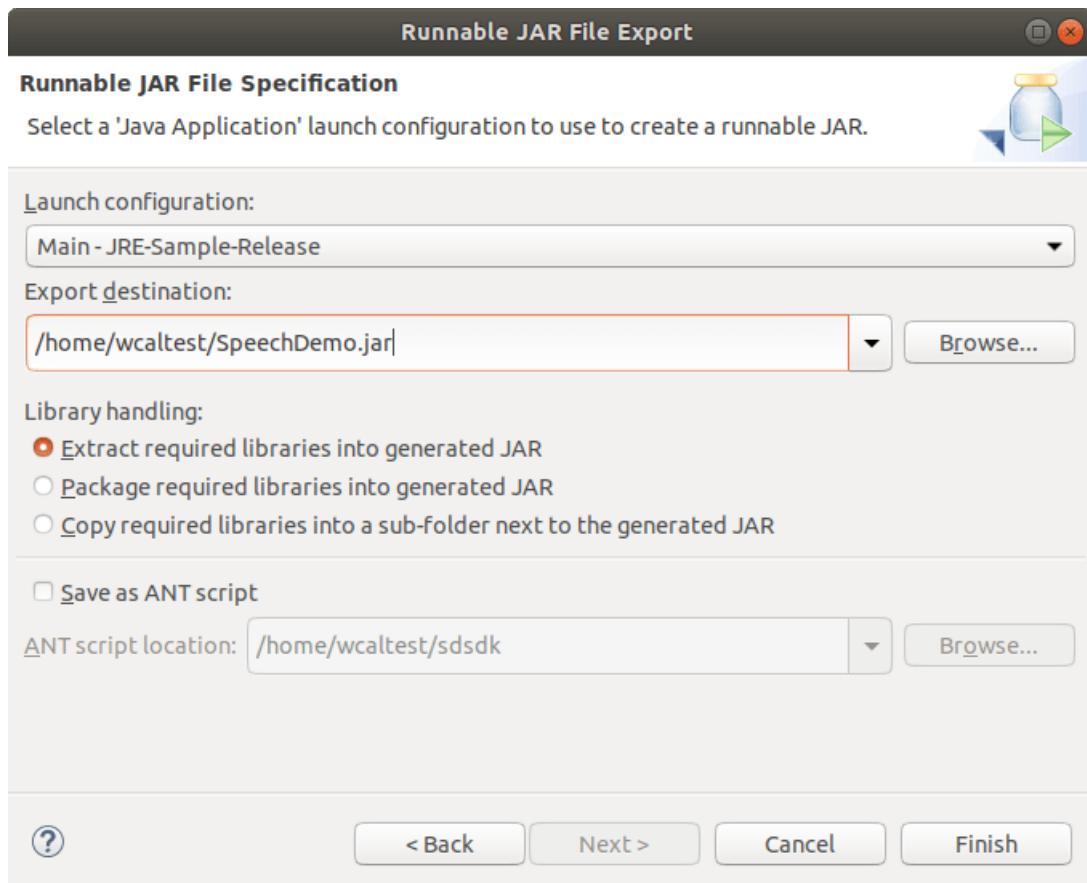


Create and run standalone the application

1. In the **Package explorer**, right-click your project. Choose **Export**.
2. The **Export** window appears. Expand **Java** and select **Runnable JAR file** and then select **Next**.



3. The **Runnable JAR File Export** window appears. Choose an **Export destination** for the application, and then select **Finish**.



4. Please put `kws.table` and `participants.properties` in the destination folder chosen above as these files are needed by the application.
5. Set the LD\_LIBRARY\_LIB to the folder containing the \*.so files

```
export LD_LIBRARY_PATH=/home/wcaltest/JRE-Sample-Release
```

6. To run the standalone application

```
java -jar SpeechDemo.jar
```

## Next steps

[Review the release notes](#)

# Quickstart: Run the Speech Devices SDK sample app on Android

12/17/2019 • 5 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to use the Speech Devices SDK for Android to build a speech-enabled product or use it as a [Conversation Transcription](#) device.

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

The source code for the sample application is included with the Speech Devices SDK. It's also [available on GitHub](#).

## Prerequisites

Before you start using the Speech Devices SDK, you'll need to:

- Follow the instructions provided with your [development kit](#) to power on the device.
- Download the latest version of the [Speech Devices SDK](#), and extract the .zip to your working directory.

### NOTE

The Android-Sample-Release.zip file includes the Android sample app and this quickstart assumes that the app is extracted to C:\SDSDK\Android-Sample-Release

- To get an [Azure subscription key for Speech service](#)
- If you plan to use the Conversation Transcription you must use a [circular microphone device](#) and this feature is currently only available for "en-US" and "zh-CN" in regions, "centralus" and "eastasia". You must have a speech key in one of those regions to use Conversation Transcription.
- If you plan to use the Speech service to identify intents (or actions) from user utterances, you'll need a [Language Understanding Service \(LUIS\)](#) subscription. To learn more about LUIS and intent recognition, see [Recognize speech intents with LUIS, C#](#).

You can [create a simple LUIS model](#) or use the sample LUIS model, LUIS-example.json. The sample LUIS model is available from the [Speech Devices SDK download site](#). To upload your model's JSON file to the [LUIS portal](#), select **Import new app**, and then select the JSON file.

- Install [Android Studio](#) and [Vysor](#) on your PC.

## Set up the device

1. Start Vysor on your computer.

Vysor

Choose a device

P818  
Serial: 4001000C0000026A

View Share  

Settings

**International Keyboard**

**Share All Devices**

**Customize Vysor** [Manage Key Bindings and Buttons](#)

**Start automatically** [Notification Prompt](#)

Status

You've used Vysor for 3 hours.  
An advertisement will be shown every 30 minutes while viewing an Android.

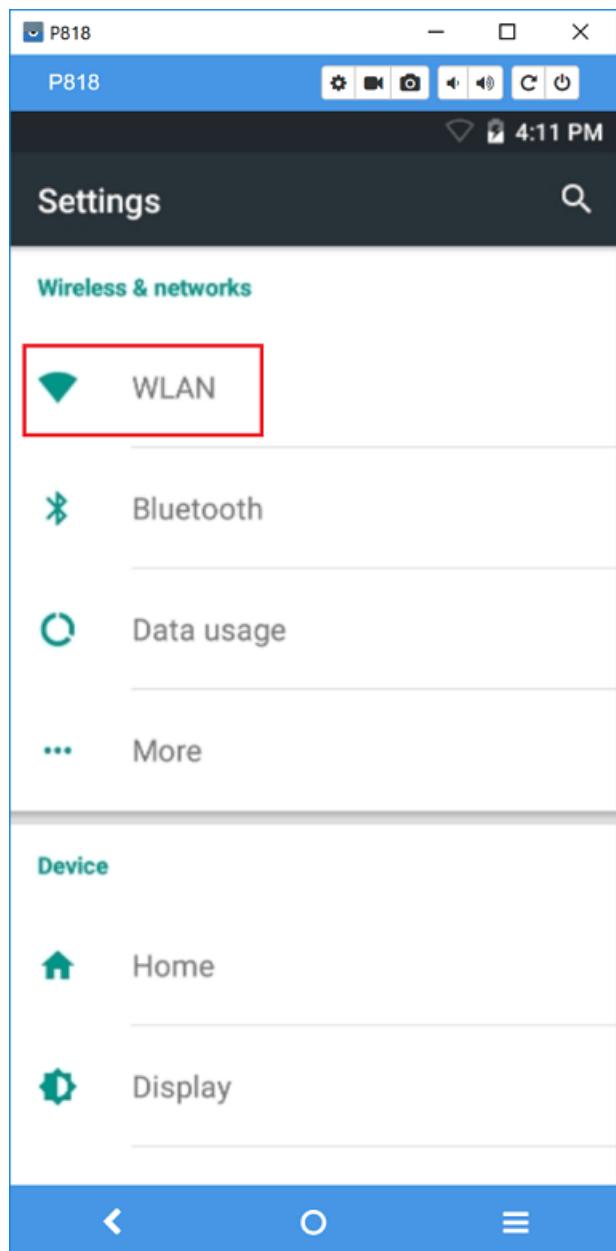
Purchase Vysor Pro to remove ads and unlock all features.

[Login](#) for offline usage and Vysor Share.  
Windows users need [ADB Drivers](#).  
Using Android SDK ADB binary.  
Vysor Version 1.9.0

[!\[\]\(4a9b8e9562b5286e35ea3cd267a6fe40\_img.jpg\)](#) [!\[\]\(922186c061ab0b31947822b0a6161910\_img.jpg\)](#) Developers Support Bug Report Manual Reload Vysor Reset Vysor

2. Your device should be listed under **Choose a device**. Select the **View** button next to the device.
3. Connect to your wireless network by selecting the folder icon, and then select **Settings > WLAN**.



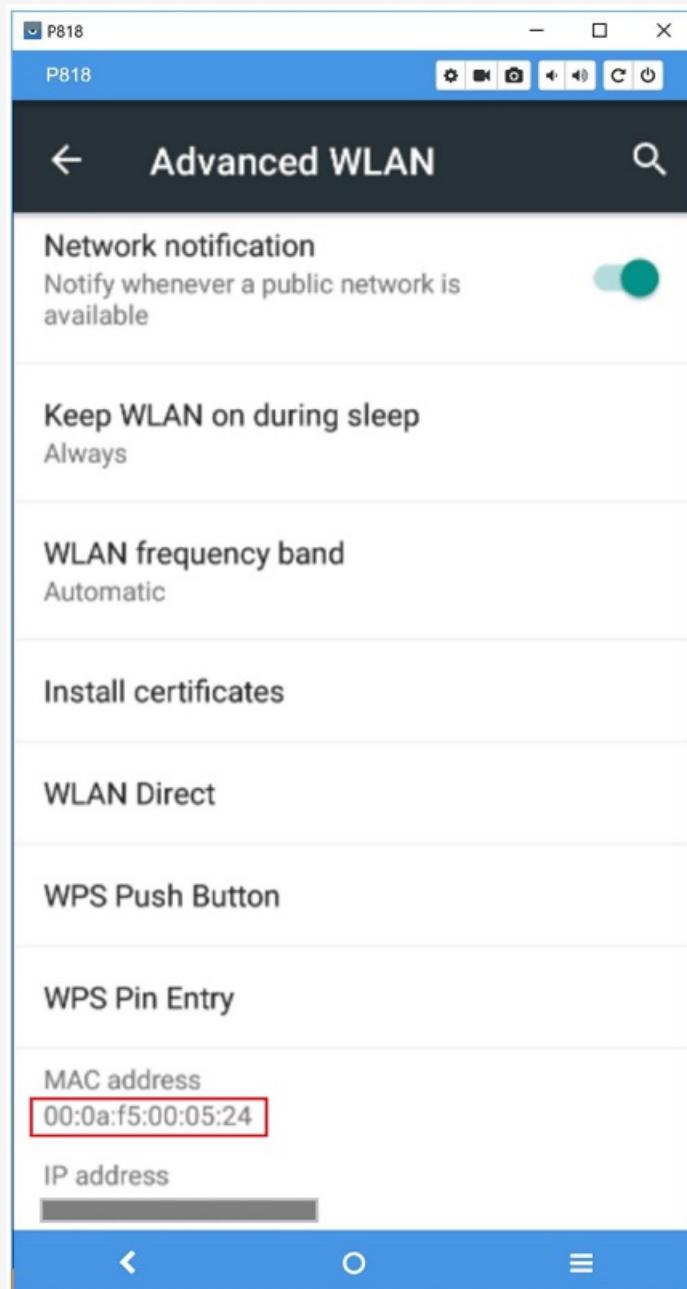
#### NOTE

If your company has policies about connecting devices to its Wi-Fi system, you need to obtain the MAC address and contact your IT department about how to connect it to your company's Wi-Fi.

To find the MAC address of the dev kit, select the file folder icon on the desktop of the dev kit.



Select **Settings**. Search for "mac address", and then select **Mac address > Advanced WLAN**. Write down the MAC address that appears near the bottom of the dialog box.

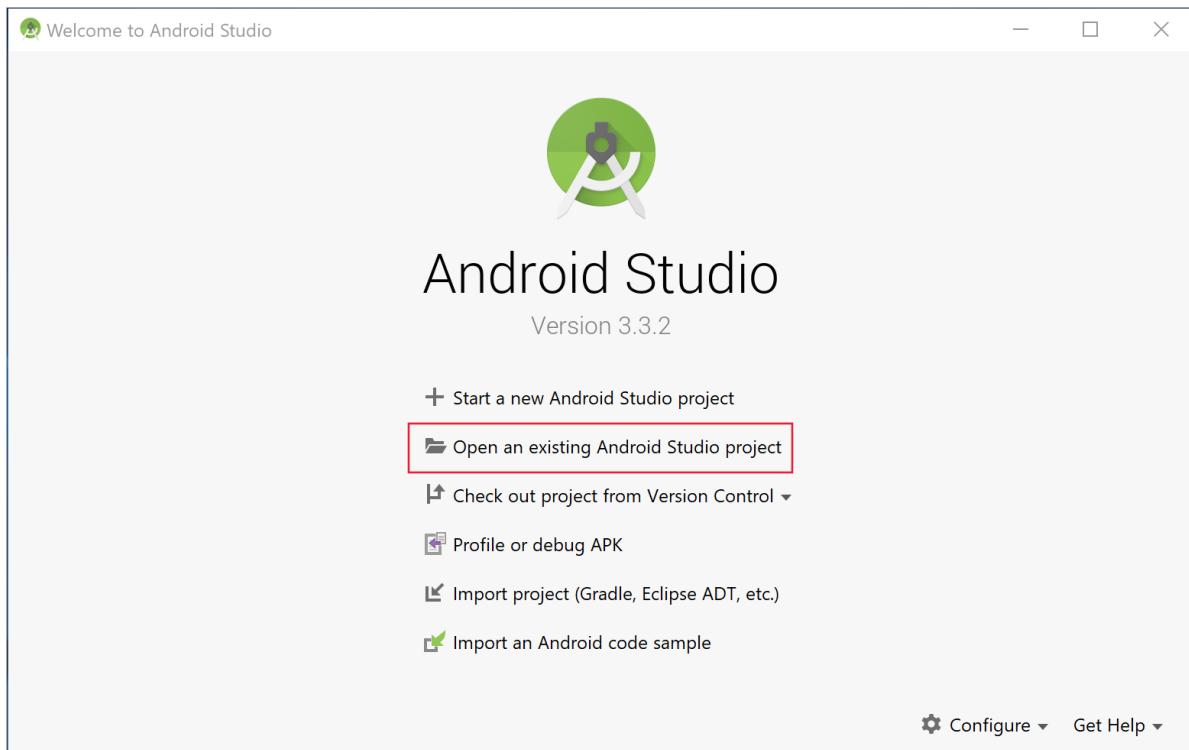


Some companies might have a time limit on how long a device can be connected to their Wi-Fi system. You might need to extend the dev kit's registration with your Wi-Fi system after a specific number of days.

## Run the sample application

To validate your development kit setup, build and install the sample application:

1. Start Android Studio.
2. Select **Open an existing Android Studio project**.



3. Go to C:\SDSDK\Android-Sample-Release\example. Select **OK** to open the example project.
4. Add your speech subscription key to the source code. If you want to try intent recognition, also add your [Language Understanding service](#) subscription key and application ID.

For speech and LUIS, your information goes into MainActivity.java:

```
// Subscription
private static String SpeechSubscriptionKey = "<enter your subscription info here>";
private static String SpeechRegion = "westus"; // You can change this if your speech region is
different.
private static String LuisSubscriptionKey = "<enter your subscription info here>";
private static String LuisRegion = "westus2"; // you can change this, if you want to test the intent,
and your LUIS region is different.
private static String LuisAppId = "<enter your LUIS AppId>";
```

If you are using conversation transcription, your speech key and region information are also needed in conversation.java:

```
private static final String CTSKey = "<Conversation Transcription Service Key>";
private static final String CTSRegion=<Conversation Transcription Service Region>;// Region may be
"centralus" or "eastasia"
```

5. The default keyword is "Computer". You can also try one of the other provided keywords, like "Machine" or "Assistant". The resource files for these alternate keywords are in the Speech Devices SDK, in the keyword folder. For example, C:\SDSDK\Android-Sample-Release\keyword\Computer contains the files used for the keyword "Computer".

#### TIP

You can also [create a custom keyword](#).

To use a new keyword, update the following two lines in `MainActivity.java`, and copy the keyword package to your app. For example, to use the keyword 'Machine' from the keyword package kws-machine.zip:

- Copy the keyword package into the folder "C:\SDSDK\Android-Sample-Release\example\app\src\main\assets\".
- Update the `MainActivity.java` with the keyword and the package name:

```
private static final String Keyword = "Machine";
private static final String KeywordModel = "kws-machine.zip" // set your own keyword package
name.
```

6. Update the following lines, which contain the microphone array geometry settings:

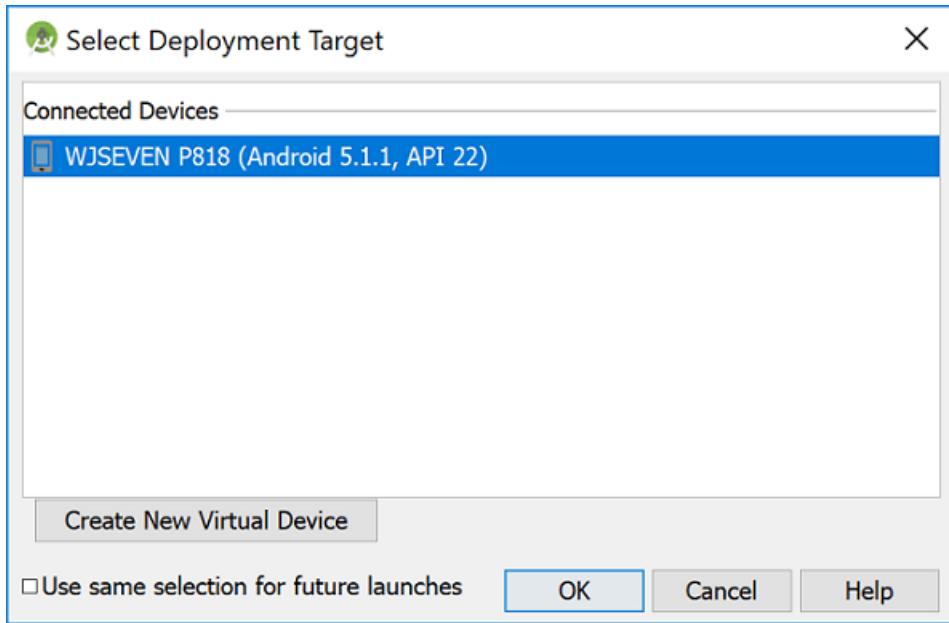
```
private static final String DeviceGeometry = "Circular6+1";
private static final String SelectedGeometry = "Circular6+1";
```

This table lists supported values:

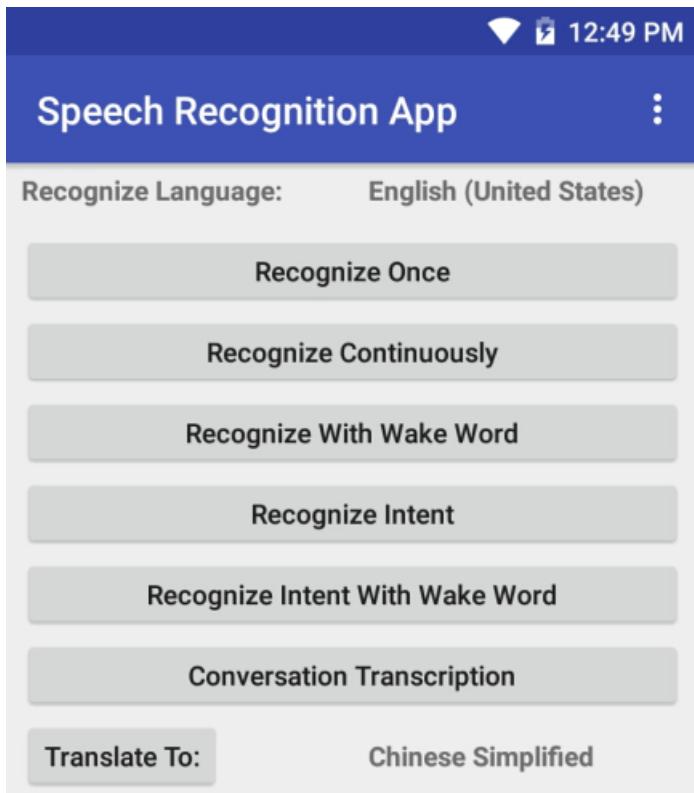
VARIABLE	MEANING	AVAILABLE VALUES
<code>DeviceGeometry</code>	Physical mic configuration	For a circular dev kit: <code>Circular6+1</code>
		For a linear dev kit: <code>Linear4</code>
<code>SelectedGeometry</code>	Software mic configuration	For a circular dev kit that uses all mics: <code>Circular6+1</code>
		For a circular dev kit that uses four mics: <code>Circular3+1</code>
		For a linear dev kit that uses all mics: <code>Linear4</code>
		For a linear dev kit that uses two mics: <code>Linear2</code>

7. To build the application, on the **Run** menu, select **Run 'app'**. The **Select Deployment Target** dialog box appears.

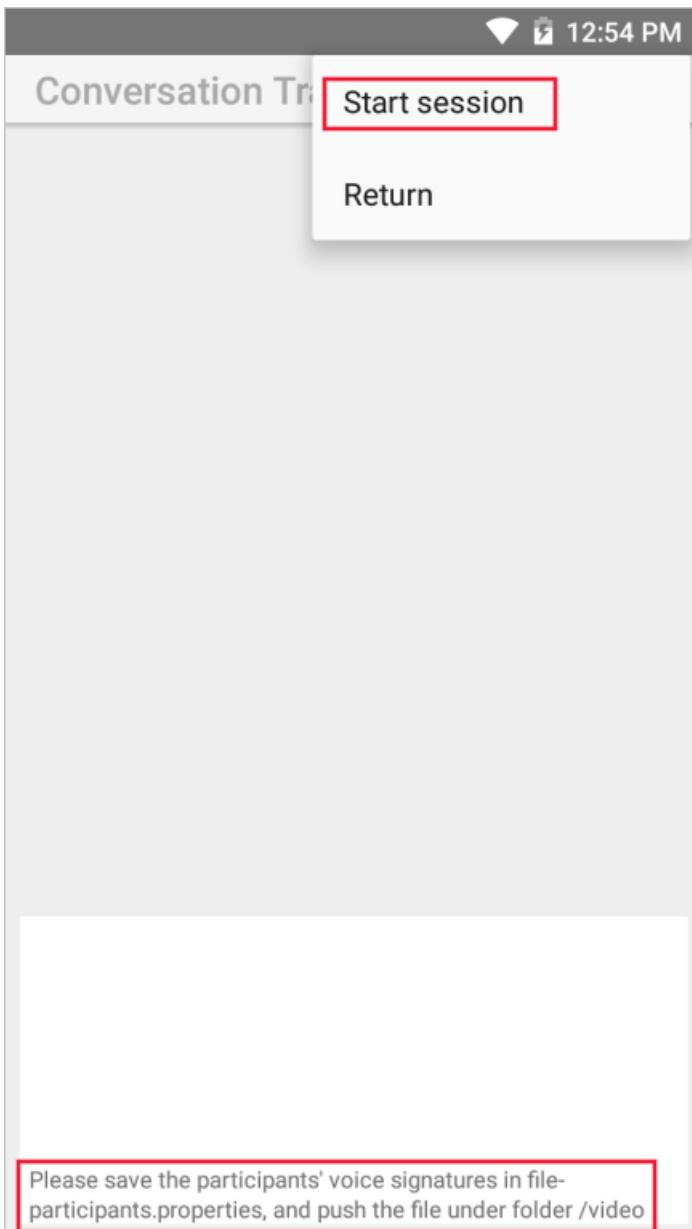
8. Select your device, and then select **OK** to deploy the application to the device.



9. The Speech Devices SDK example application starts and displays the following options:



10. Try the new Conversation Transcription demo. Start transcribing with 'Start Session'. By default everyone is a guest. However, if you have participant's voice signatures they can be put into a file `/video/participants.properties` on the device. To generate the voice signature, look at [Transcribe conversations \(SDK\)](#).



## 11. Experiment!

## Troubleshooting

If you cannot connect to the Speech Device. Type the following command in a Command Prompt window. It will return a list of devices:

```
adb devices
```

### NOTE

This command uses the Android Debug Bridge, `adb.exe`, which is part of the Android Studio installation. This tool is located in `C:\Users[user name]\AppData\Local\Android\Sdk\platform-tools`. You can add this directory to your path to make it more convenient to invoke `adb`. Otherwise, you must specify the full path to your installation of `adb.exe` in every command that invokes `adb`.

If you see an error `no devices/emulators found` then check your USB cable is connected and ensure a high quality cable is used.

## Next steps

[Review the release notes](#)

# Troubleshoot the Speech Devices SDK

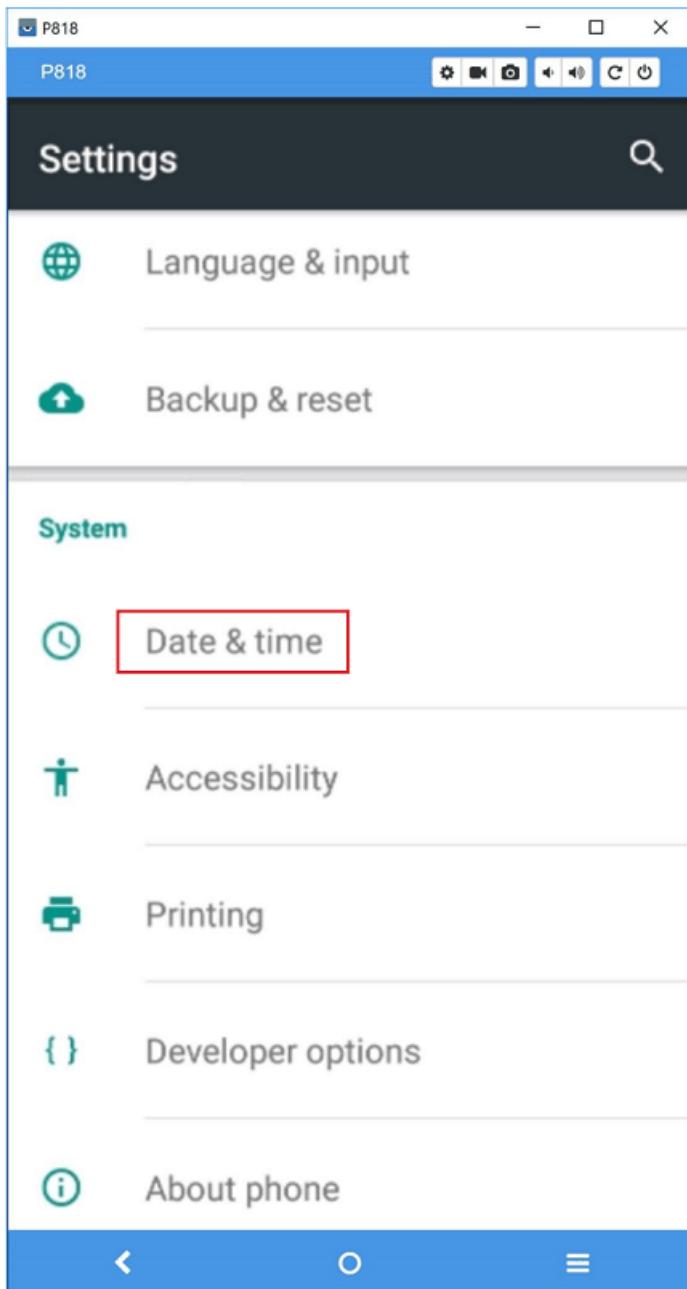
12/4/2019 • 2 minutes to read • [Edit Online](#)

This article provides information to help you solve issues you might encounter when you use the Speech Devices SDK.

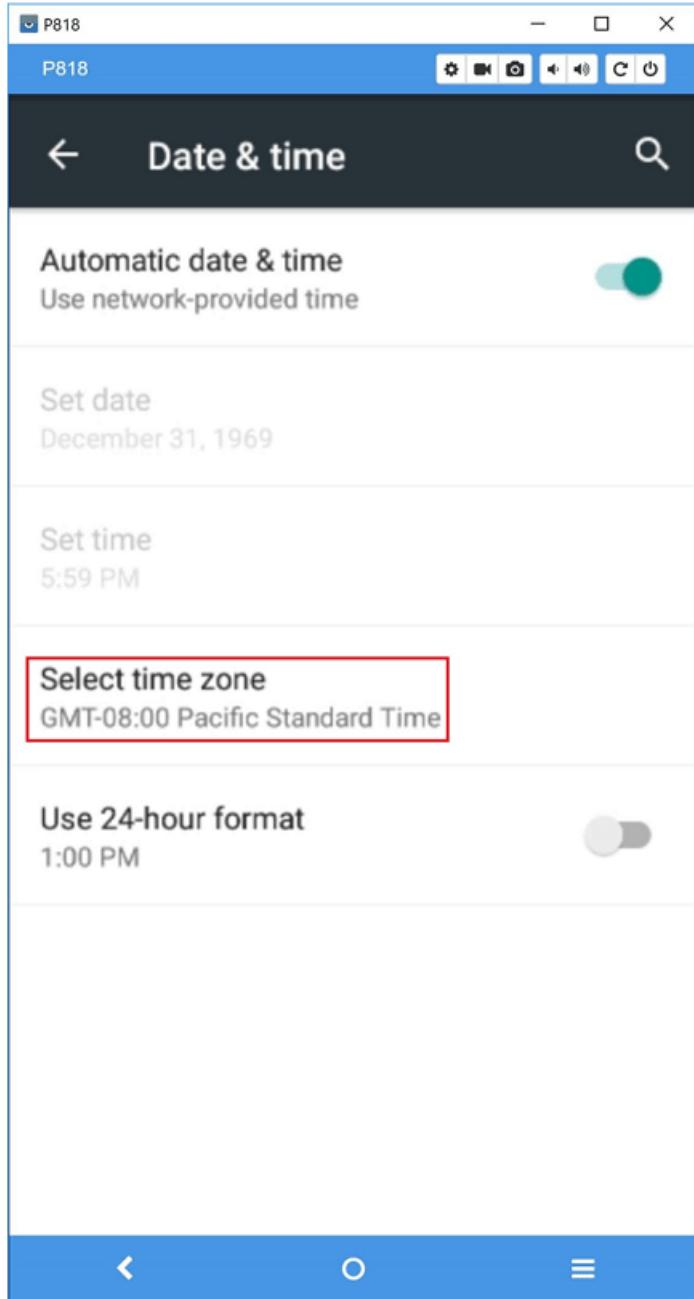
## Certificate failures

If you get certificate failures when using the Speech service, make sure that your device has the correct date and time:

1. Go to **Settings**. Under **System**, select **Date & time**.



2. Keep the **Automatic date & time** option selected. Under **Select time zone**, select your current time zone.



When you see that the dev kit's time matches the time on your PC, the dev kit is connected to the internet.

## Next steps

- [Review the release notes](#)

# Release notes: Speech Devices SDK

12/11/2019 • 2 minutes to read • [Edit Online](#)

The following sections list changes in the most recent releases.

## Speech Devices SDK 1.7.0:

- Linux ARM is now supported.
- Initial binaries for Roobo v2 are provided (Linux ARM64).
- Windows users can use `AudioConfig.fromDefaultMicrophoneInput()` or `AudioConfig.fromMicrophoneInput(deviceName)` to specify the microphone to be used.
- The library size has been optimized.
- Support for multi-turn recognition using the same speech/intent recognizer object.
- Fix occasional hang that would occur while stopping recognition.
- Sample apps now contain a sample participants.properties file to demonstrate the format of the file.
- Updated the [Speech SDK](#) component to version 1.7.0. For more information, see its [release notes](#).

## Speech Devices SDK 1.6.0:

- Support [Azure Kinect DK](#) on Windows and Linux with common [sample application](#)
- Updated the [Speech SDK](#) component to version 1.6.0. For more information, see its [release notes](#).

## Speech Devices SDK 1.5.1:

- Include [Conversation Transcription](#) in the sample app.
- Updated the [Speech SDK](#) component to version 1.5.1. For more information, see its [release notes](#).

## Speech Devices SDK 1.5.0: 2019-May release

- Speech Devices SDK is now GA and no longer a gated preview.
- Updated the [Speech SDK](#) component to version 1.5.0. For more information, see its [release notes](#).
- New keyword technology brings significant quality improvements, see [Breaking Changes](#).
- New audio processing pipeline for improved far-field recognition.

### **Breaking changes**

- Due to the new keyword technology all keywords must be re-created at our improved keyword portal. To fully remove old keywords from the device uninstall the old app.
  - adb uninstall com.microsoft.cognitiveservices.speech.samples.sdsdkstarterapp

## Speech Devices SDK 1.4.0: 2019-Apr release

- Updated the [Speech SDK](#) component to version 1.4.0. For more information, see its [release notes](#).

## Speech Devices SDK 1.3.1: 2019-Mar release

- Updated the [Speech SDK](#) component to version 1.3.1. For more information, see its [release notes](#).
- Updated keyword handling, see [Breaking Changes](#).
- Sample application adds choice of language for both speech recognition and translation.

## **Breaking changes**

- [Installing a keyword](#) has been simplified, it is now part of the app and does not need separate installation on the device.
- The keyword recognition has changed, and two events are supported.
  - `RecognizingKeyword`, indicates the speech result contains (unverified) keyword text.
  - `RecognizedKeyword`, indicates that keyword recognition completed recognizing the given keyword.

## Speech Devices SDK 1.1.0: 2018-Nov release

- Updated the [Speech SDK](#) component to version 1.1.0. For more information, see its [release notes](#).
- Far Field Speech recognition accuracy has been improved with our enhanced audio processing algorithm.
- Sample application added Chinese speech recognition support.

## Speech Devices SDK 1.0.1: 2018-Oct release

- Updated the [Speech SDK](#) component to version 1.0.1. For more information, see its [release notes](#).
- Speech recognition accuracy will be improved with our improved audio processing algorithm
- One continuous recognition audio session bug is fixed.

## **Breaking changes**

- With this release a number of breaking changes are introduced. Please check [this page](#) for details relating to the APIs.
- The KWS model files are not compatible with Speech Devices SDK 1.0.1. The existing keyword files will be deleted after the new keyword files are written to the device.

## Speech Devices SDK 0.5.0: 2018-Aug release

- Improved the accuracy of speech recognition by fixing a bug in the audio processing code.
- Updated the [Speech SDK](#) component to version 0.5.0. For more information, see its [release notes](#).

## Speech Devices SDK 0.2.12733: 2018-May release

The first public preview release of the Cognitive Services Speech Devices SDK.

# Migrate from Bing Speech to the Speech service

12/4/2019 • 4 minutes to read • [Edit Online](#)

Use this article to migrate your applications from the Bing Speech API to the Speech service.

This article outlines the differences between the Bing Speech APIs and the Speech service, and suggests strategies for migrating your applications. Your Bing Speech API subscription key won't work with the Speech service; you'll need a new Speech service subscription.

A single Speech service subscription key grants access to the following features. Each is metered separately, so you're charged only for the features you use.

- [Speech-to-text](#)
- [Custom speech-to-text](#)
- [Text-to-speech](#)
- [Custom text-to-speech voices](#)
- [Speech translation](#) (does not include [Text translation](#))

The [Speech SDK](#) is a functional replacement for the Bing Speech client libraries, but uses a different API.

## Comparison of features

The Speech service is largely similar to Bing Speech, with the following differences.

FEATURE	BING SPEECH	SPEECH SERVICE	DETAILS
C++ SDK	✗	✓✗	Speech service supports Windows and Linux.
Java SDK	✓✗	✓✗	Speech service supports Android and Speech Devices.
C# SDK	✓✗	✓✗	Speech service supports Windows 10, Universal Windows Platform (UWP), and .NET Standard 2.0.
Continuous speech recognition	10 minutes	Unlimited (with SDK)	Both Bing Speech and Speech service WebSockets protocols support up to 10 minutes per call. However, the Speech SDK automatically reconnects on timeout or disconnect.
Partial or interim results	✓✗	✓✗	With WebSockets protocol or SDK.
Custom speech models	✓✗	✓✗	Bing Speech requires a separate Custom Speech subscription.

FEATURE	BING SPEECH	SPEECH SERVICE	DETAILS
Custom voice fonts	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	Bing Speech requires a separate Custom Voice subscription.
24-KHz voices	<input type="checkbox"/>	✓ <input type="checkbox"/>	
Speech intent recognition	Requires separate LUIS API call	Integrated (with SDK)	You can use a LUIS key with the Speech service.
Simple intent recognition	<input type="checkbox"/>	✓ <input type="checkbox"/>	
Batch transcription of long audio files	<input type="checkbox"/>	✓ <input type="checkbox"/>	
Recognition mode	Manual via endpoint URI	Automatic	Recognition mode is not available in the Speech service.
Endpoint locality	Global	Regional	Regional endpoints improve latency.
REST APIs	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	The Speech service REST APIs are compatible with Bing Speech (different endpoint). REST APIs support text-to-speech and limited speech-to-text functionality.
WebSockets protocols	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	The Speech service WebSockets API is compatible with Bing Speech (different endpoint). Migrate to the Speech SDK if possible, to simplify your code.
Service-to-service API calls	✓ <input type="checkbox"/>	<input type="checkbox"/>	Provided in Bing Speech via the C# Service Library.
Open source SDK	✓ <input type="checkbox"/>	<input type="checkbox"/>	

The Speech service uses a time-based pricing model (rather than a transaction-based model). See [Speech service pricing](#) for details.

## Migration strategies

If you or your organization have applications in development or production that use a Bing Speech API, you should update them to use the Speech service as soon as possible. See the [Speech service documentation](#) for available SDKs, code samples, and tutorials.

The Speech service [REST APIs](#) are compatible with the Bing Speech APIs. If you're currently using the Bing Speech REST APIs, you need only change the REST endpoint, and switch to a Speech service subscription key.

The Speech service WebSockets protocols are also compatible with those used by Bing Speech. We recommend

that for new development, you use the Speech SDK rather than WebSockets. It's a good idea to migrate existing code to the SDK as well. However, as with the REST APIs, existing code that uses Bing Speech via WebSockets requires only a change in endpoint and an updated key.

If you're using a Bing Speech client library for a specific programming language, migrating to the [Speech SDK](#) requires changes to your application, because the API is different. The Speech SDK can make your code simpler, while also giving you access to new features.

Currently, the Speech SDK supports C# ([details here](#)), Java (Android and custom devices), Objective C (iOS), C++ (Windows and Linux), and JavaScript. APIs on all platforms are similar, easing multi-platform development.

The Speech service doesn't offer a global endpoint. Determine if your application functions efficiently when it uses a single regional endpoint for all of its traffic. If not, use geolocation to determine the most efficient endpoint. You need a separate Speech service subscription in each region you use.

If your application uses long-lived connections and can't use an available SDK, you can use a WebSockets connection. Manage the 10-minute timeout limit by reconnecting at the appropriate times.

To get started with the Speech SDK:

1. Download the [Speech SDK](#).
2. Work through the Speech service [quickstart guides](#) and [tutorials](#). Also look at the [code samples](#) to get experience with the new APIs.
3. Update your application to use the Speech service.

## Support

Bing Speech customers should contact customer support by opening a [support ticket](#). You can also contact us if your support need requires a [technical support plan](#).

For Speech service, SDK, and API support, visit the Speech service [support page](#).

## Next steps

- [Try out Speech service for free](#)
- [Quickstart: Recognize speech in a UWP app using the Speech SDK](#)

## See also

- [Speech service release notes](#)
- [What is the Speech service](#)
- [Speech service and Speech SDK documentation](#)

# Migrate from the Custom Speech Service to the Speech service

12/4/2019 • 2 minutes to read • [Edit Online](#)

Use this article to migrate your applications from the Custom Speech Service to the Speech service.

The Custom Speech Service is now part of the Speech service. Switch to the Speech service to benefit from the latest quality and feature updates.

## Migration for new customers

The pricing model is simpler, using an hour-based pricing model for the Speech service.

1. Create an Azure resource in each region where your application is available. The Azure resource name is **Speech**. You can use a single Azure resource for the following services in the same region, instead of creating separate resources:
  - Speech-to-text
  - Custom speech-to-text
  - Text-to-speech
  - Speech translation
2. Download the [Speech SDK](#).
3. Follow the quickstart guides and SDK samples to use the correct APIs. If you use the REST APIs, you also need to use the correct endpoints and resource keys.
4. Update the client application to use the Speech service and APIs.

## Migration for existing customers

Migrate your existing resource keys to the Speech service on the Speech service portal. Use the following steps:

### NOTE

Resource keys can only be migrated within the same region.

1. Sign in to the [cris.ai](#) portal, and select the subscription in the top right menu.
2. Select **Migrate selected subscription**.
3. Enter the subscription key in the text box, and select **Migrate**.

## Next steps

- [Try out Speech service for free](#).
- Learn [speech to text](#) concepts.

## See also

- [What is the Speech service](#)
- [Speech service and Speech SDK documentation](#)



# Migrate from the Translator Speech API to the Speech service

12/4/2019 • 2 minutes to read • [Edit Online](#)

Use this article to migrate your applications from the Microsoft Translator Speech API to the [Speech service](#). This guide outlines the differences between the Translator Speech API and Speech service, and suggests strategies for migrating your applications.

## NOTE

Your Translator Speech API subscription key won't be accepted by the Speech service. You'll need to create a new Speech service subscription.

## Comparison of features

FEATURE	TRANSLATOR SPEECH API	SPEECH SERVICE	DETAILS
Translation to text	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	
Translation to speech	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	
Global endpoint	✓ <input type="checkbox"/>	<input type="checkbox"/>	The Speech service doesn't offer a global endpoint. A global endpoint can automatically direct traffic to the nearest regional endpoint, decreasing latency in your application.
Regional endpoints	<input type="checkbox"/>	✓ <input type="checkbox"/>	
Connection time limit	90 minutes	Unlimited with the SDK. 10 minutes with a WebSockets connection.	
Auth key in header	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	
Multiple languages translated in a single request	<input type="checkbox"/>	✓ <input type="checkbox"/>	
SDKs available	<input type="checkbox"/>	✓ <input type="checkbox"/>	See the <a href="#">Speech service documentation</a> for available SDKs.
WebSockets connections	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	

FEATURE	TRANSLATOR SPEECH API	SPEECH SERVICE	DETAILS
Languages API	✓ <input type="checkbox"/>	<input type="checkbox"/>	The Speech service supports the same range of languages described in the <a href="#">Translator API languages reference</a> article.
Profanity Filter and Marker	<input type="checkbox"/>	✓ <input type="checkbox"/>	
.WAV/PCM as input	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	
Other file types as input	<input type="checkbox"/>	<input type="checkbox"/>	
Partial results	✓ <input type="checkbox"/>	✓ <input type="checkbox"/>	
Timing info	✓ <input type="checkbox"/>	<input type="checkbox"/>	
Correlation ID	✓ <input type="checkbox"/>	<input type="checkbox"/>	
Custom speech models	<input type="checkbox"/>	✓ <input type="checkbox"/>	The Speech service offers custom speech models that enable you to customize speech recognition to your organization's unique vocabulary.
Custom translation models	<input type="checkbox"/>	✓ <input type="checkbox"/>	Subscribing to the Microsoft Text Translation API enables you to use <a href="#">Custom Translator</a> to use your own data for more accurate translations.

## Migration strategies

If you or your organization have applications in development or production that use the Translator Speech API, you should update them to use the Speech service. See the [Speech service](#) documentation for available SDKs, code samples, and tutorials. Consider the following when you are migrating:

- The Speech service doesn't offer a global endpoint. Determine if your application functions efficiently when it uses a single regional endpoint for all of its traffic. If not, use geolocation to determine the most efficient endpoint.
- If your application uses long-lived connections and can't use the available SDKs, you can use a WebSockets connection. Manage the 10-minute timeout limit by reconnecting at the appropriate times.
- If your application uses the Translator Text API and Translator Speech API to enable custom translation models, you can add Category IDs directly by using the Speech service.
- Unlike the Translator Speech API, the Speech service can complete translations into multiple languages in a single request.

## Next steps

- [Try out Speech service for free](#)

- [Quickstart: Recognize speech in a UWP app using the Speech SDK](#)

## See also

- [What is the Speech service](#)
- [Speech service and Speech SDK documentation](#)

# Support and help options

12/6/2019 • 2 minutes to read • [Edit Online](#)

Are you just starting to explore the functionality of the Speech service? Are you implementing a new feature to your application? Here are suggestions about where you can get help as a developer.

- Stay informed about new developments in *Azure Cognitive Services*, or find the latest news related to the *Speech service*.
- Release notes contain information for all releases.
- Search to see if your issue was discussed by the community, or if existing documentation for the feature you want to implement already exists.
- If you can't find a satisfactory answer, ask a question on *Stack Overflow*.
- If you find an issue with one of the samples on GitHub, raise a *GitHub* issue.
- Search for a solution in the *UserVoice forum*.

## Stay informed

News about Cognitive Services is collected in the [Cognitive Services blog](#). For the latest information about the Speech service, track the [Speech service blog](#).

## Release notes

The [release notes](#) are updated as soon as a new release is available. The notes contain information about new features, improvements, and bug fixes.

## Search

You might find the answer you need in the documentation, the samples, or answers to [Stack Overflow](#) questions or in the samples.

### Scoped Search

For faster results, scope your search to Stack Overflow, the documentation, and code samples by using the following query on your [favorite search engine](#):

```
{Your Search Terms} (site:stackoverflow.com OR site:docs.microsoft.com OR site:github.com/azure-samples)
```

Where *{Your Search Terms}* is your search keywords.

## Create an Azure support request

Azure customers can create and manage support requests in the Azure portal.

- [Azure portal](#)
- [Azure portal for the United States government](#)

## Post a question to Stack Overflow

Stack Overflow is the preferred channel for development-related questions. It's where members of the community and Microsoft team members are directly involved in helping you solve your problems.

If you can't find an answer to your problem via search, submit a new question to Stack Overflow by using tags [\[microsoft-cognitive\]\[speech\]](#).

#### TIP

The following posts from Stack Overflow contain tips on how to form questions and add source code. Following these guidelines might help increase the chances that community members assess and respond to your question quickly:

- [How do I ask a good question?](#)
- [How to create a Minimal, Complete, and Verifiable example](#)

## Create a GitHub issue

Samples are often posted as open source. For questions and issues, create an *issue* in the respective GitHub repository. You can submit a pull request, too. The following list contains links to the sample repositories:

- [Speech SDK](#)
- [Speech Devices SDK](#)

You can create a bug report, feature request, or ask a general question and share best practices. For bug reports, please follow the provided template:

### Describe the bug

A clear and concise description of what the bug is.

### To Reproduce

Steps to reproduce the behavior:

1. ...
2. ...

### Expected behavior

A clear and concise description of what you expected to happen.

### Version of the Cognitive Services Speech SDK

Which version of the SDK are you using.

### Platform, Operating System, and Programming Language

- OS: [e.g. Windows, Linux, Android, iOS, ...] - please be specific
- Hardware - x64, x86, ARM, ...
- Browser [e.g. Chrome, Safari] (if applicable)- please be specific

### Additional context

- Error messages, log information, stack trace, ...
- If you report an error for a specific service interaction, report the SessionId and time (incl. timezone) of the reported incidents. The SessionId is reported in all call-backs/events you receive.
- Any other additional information

## UserVoice forum

Share your ideas for making Cognitive Services and the accompanying APIs work better for the applications you develop. Use our growing Knowledge Base to find answers to common questions:



# Speech service supported regions

12/4/2019 • 3 minutes to read • [Edit Online](#)

The Speech service allows your application to convert audio to text, perform speech translation, and convert text to speech. The service is available in multiple regions with unique endpoints for the Speech SDK and REST APIs.

Make sure that you use the endpoint that matches the region for your subscription.

## Speech SDK

In the [Speech SDK](#), regions are specified as a string (for example, as a parameter to `SpeechConfig.FromSubscription` in the Speech SDK for C#).

### Speech-to-text, text-to-speech, and translation

The Speech SDK is available in these regions for **speech recognition**, **text-to-speech**, and **translation**:

REGION	SPEECH SDK PARAMETER	SPEECH CUSTOMIZATION PORTAL
West US	<code>westus</code>	<a href="https://westus.cris.ai">https://westus.cris.ai</a>
West US 2	<code>westus2</code>	<a href="https://westus2.cris.ai">https://westus2.cris.ai</a>
East US	<code>eastus</code>	<a href="https://eastus.cris.ai">https://eastus.cris.ai</a>
East US 2	<code>eastus2</code>	<a href="https://eastus2.cris.ai">https://eastus2.cris.ai</a>
Central US	<code>centralus</code>	<a href="https://centralus.cris.ai">https://centralus.cris.ai</a>
North Central US	<code>northcentralus</code>	<a href="https://northcentralus.cris.ai">https://northcentralus.cris.ai</a>
South Central US	<code>southcentralus</code>	<a href="https://southcentralus.cris.ai">https://southcentralus.cris.ai</a>
Central India	<code>centralindia</code>	<a href="https://centralindia.cris.ai">https://centralindia.cris.ai</a>
East Asia	<code>eastasia</code>	<a href="https://eastasia.cris.ai">https://eastasia.cris.ai</a>
Southeast Asia	<code>southeastasia</code>	<a href="https://southeastasia.cris.ai">https://southeastasia.cris.ai</a>
Japan East	<code>japaneast</code>	<a href="https://japaneast.cris.ai">https://japaneast.cris.ai</a>
Korea Central	<code>koreacentral</code>	<a href="https://koreacentral.cris.ai">https://koreacentral.cris.ai</a>
Australia East	<code>australiaeast</code>	<a href="https://australiaeast.cris.ai">https://australiaeast.cris.ai</a>
Canada Central	<code>canadacentral</code>	<a href="https://canadacentral.cris.ai">https://canadacentral.cris.ai</a>
North Europe	<code>northeurope</code>	<a href="https://northeurope.cris.ai">https://northeurope.cris.ai</a>
West Europe	<code>westeurope</code>	<a href="https://westeurope.cris.ai">https://westeurope.cris.ai</a>
UK South	<code>uksouth</code>	<a href="https://uksouth.cris.ai">https://uksouth.cris.ai</a>
France Central	<code>francecentral</code>	<a href="https://francecentral.cris.ai">https://francecentral.cris.ai</a>

### Intent recognition

Available regions for **intent recognition** via the Speech SDK are the following:

GLOBAL REGION	REGION	SPEECH SDK PARAMETER
Asia	East Asia	<code>eastasia</code>
Asia	Southeast Asia	<code>southeastasia</code>
Australia	Australia East	<code>australiaeast</code>
Europe	North Europe	<code>northeurope</code>
Europe	West Europe	<code>westeurope</code>

GLOBAL REGION	REGION	SPEECH SDK PARAMETER
North America	East US	<code>eastus</code>
North America	East US 2	<code>eastus2</code>
North America	South Central US	<code>southcentralus</code>
North America	West Central US	<code>westcentralus</code>
North America	West US	<code>westus</code>
North America	West US 2	<code>westus2</code>
South America	Brazil South	<code>brazilsouth</code>

This is a subset of the publishing regions supported by the [Language Understanding service \(LUIS\)](#).

### Voice assistants

The [Speech SDK](#) supports **voice assistant** capabilities in these regions:

REGION	SPEECH SDK PARAMETER
West US	<code>westus</code>
West US 2	<code>westus2</code>
East US	<code>eastus</code>
East US 2	<code>eastus2</code>
West Europe	<code>westeurope</code>
North Europe	<code>northeurope</code>
Southeast Asia	<code>southeastasia</code>

## REST APIs

The Speech service also exposes REST endpoints for speech-to-text and text-to-speech requests.

### Speech-to-text

For speech-to-text reference documentation, see [Speech-to-text REST API](#).

REGION	ENDPOINT
Australia East	<a href="https://australiaeast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://australiaeast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Canada Central	<a href="https://canadacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://canadacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Central US	<a href="https://centralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://centralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
East Asia	<a href="https://eastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://eastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
East US	<a href="https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://eastus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
East US 2	<a href="https://eastus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://eastus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
France Central	<a href="https://francecentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://francecentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
India Central	<a href="https://centralindia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://centralindia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Japan East	<a href="https://japaneast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://japaneast.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Korea Central	<a href="https://koreacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://koreacentral.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
North Central US	<a href="https://northcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://northcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
North Europe	<a href="https://northeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://northeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>

REGION	ENDPOINT
South Central US	<a href="https://southcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://southcentralus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
Southeast Asia	<a href="https://southeastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://southeastasia.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
UK South	<a href="https://uksouth.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://uksouth.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
West Europe	<a href="https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://westeurope.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
West US	<a href="https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>
West US 2	<a href="https://westus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1">https://westus2.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1</a>

#### NOTE

The language parameter must be appended to the URL to avoid receiving an 4xx HTTP error. For example, the language set to US English using the West US endpoint is: <https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US>.

### Text-to-speech

For text-to-speech reference documentation, see [Text-to-speech REST API](#).

### Standard and neural voices

Use this table to determine availability of standard and neural voices by region/endpoint:

REGION	ENDPOINT	STANDARD VOICES	NEURAL VOICES
Australia East	<a href="https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1">https://australiaeast.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
Canada Central	<a href="https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1">https://canadacentral.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
Central US	<a href="https://centralus.tts.speech.microsoft.com/cognitiveservices/v1">https://centralus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
East Asia	<a href="https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1">https://eastasia.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
East US	<a href="https://eastus.tts.speech.microsoft.com/cognitiveservices/v1">https://eastus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
East US 2	<a href="https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1">https://eastus2.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
France Central	<a href="https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1">https://francecentral.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
India Central	<a href="https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1">https://centralindia.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
Japan East	<a href="https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1">https://japaneast.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
Korea Central	<a href="https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1">https://koreacentral.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
North Central US	<a href="https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1">https://northcentralus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
North Europe	<a href="https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1">https://northeurope.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
South Central US	<a href="https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1">https://southcentralus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
Southeast Asia	<a href="https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1">https://southeastasia.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
UK South	<a href="https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1">https://uksouth.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
West Europe	<a href="https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1">https://westeurope.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes
West US	<a href="https://westus.tts.speech.microsoft.com/cognitiveservices/v1">https://westus.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	No
West US 2	<a href="https://westus2.tts.speech.microsoft.com/cognitiveservices/v1">https://westus2.tts.speech.microsoft.com/cognitiveservices/v1</a>	Yes	Yes

### Custom voices

If you've created a custom voice font, use the endpoint that you've created. You can also use the endpoints listed below, replacing the `{deploymentId}` with the deployment ID for your voice model.

REGION	ENDPOINT
Australia East	<a href="https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://australiaeast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Canada Central	<a href="https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://canadacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Central US	<a href="https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://centralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
East Asia	<a href="https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://eastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
East US	<a href="https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://eastus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
East US 2	<a href="https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://eastus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
France Central	<a href="https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://francecentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
India Central	<a href="https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://centralindia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Japan East	<a href="https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://japaneast.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Korea Central	<a href="https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://koreacentral.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
North Central US	<a href="https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://northcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
North Europe	<a href="https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://northeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
South Central US	<a href="https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://southcentralus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
Southeast Asia	<a href="https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://southeastasia.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
UK South	<a href="https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://uksouth.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
West Europe	<a href="https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://westeurope.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
West US	<a href="https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://westus.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>
West US 2	<a href="https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}">https://westus2.voice.speech.microsoft.com/cognitiveservices/v1?deploymentId={deploymentId}</a>